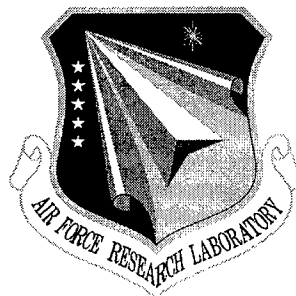


AFRL-IF-RS-TR-1998-39
Final Technical Report
May 1998



OPTIMIZATION TECHNIQUES FOR TRUSTED SEMANTIC INTEROPERATION

SRI International

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. B401

19980708 021

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

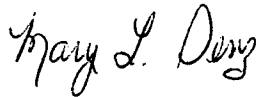
AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

[DTIC QUALITY INSPECTED 1]

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1998-39 has been reviewed and is approved for publication.

APPROVED:



MARY L. DENZ
Project Engineer

FOR THE DIRECTOR:



WARREN H. DEBANY, JR.
Technical Advisor, Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

OPTIMIZATION TECHNIQUES FOR TRUSTED SEMANTIC INTEROPERATION

Steven Dawsen

Contractor: SRI International

Contract Number: F30602-94-C-0198

Effective Date of Contract: 31 August 1994

Contract Expiration Date: 30 October 1997

Program Code Number: 7E20

Short Title of Work: Optimization Techniques for Trusted
Semantic Interoperation

Period of Work Covered: Aug 94 - Oct 97

Principal Investigator: Steven Dawsen

Phone: (415) 859-5390

AFRL Project Engineer: Mary L. Denz

Phone: (315) 330-2030

Approved for public release; distribution unlimited.

This research was supported by the Advanced Research Projects
Agency of the Department of Defense and was monitored by
Mary L. Denz, AFRL/IFGB, 525 Brooks Rd, Rome, NY 13441-4505.

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 | |
|---|---|--|---|--|
| <small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small> | | | | |
| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE May 1998 | 3. REPORT TYPE AND DATES COVERED Final Aug 94 - Oct 97 | | |
| 4. TITLE AND SUBTITLE OPTIMIZATION TECHNIQUES FOR TRUSTED SEMANTIC INTEROPERATION | | 5. FUNDING NUMBERS C - F30602-94-C-0198 PE - 62301E, 33140F PR - B401 TA - 00 WU - 01 | | |
| 6. AUTHOR(S) Steven Dawsen | | 8. PERFORMING ORGANIZATION REPORT NUMBER N/A | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Avenue Menlo Park CA 94025-3493 | | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-1998-39 | | |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency Air Force Research Laboratory/IFGB 3701 North Fairfax Drive 525 Brooks Road Arlington VA 22203-1714 Rome NY 13441-4505 | | | | |
| 11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Mary L. Denz/IFGB/(315) 330-2030 | | | | |
| 12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited. | | | 12b. DISTRIBUTION CODE | |
| 13. ABSTRACT (Maximum 200 words) SRI International has completed a research program on the development of techniques and tools for efficient, trusted interoperation of autonomous heterogeneous information sources. This program has produced novel techniques for integration of heterogeneous databases into large applications, efficient algorithms for optimization and security of semantic interoperaiton, and a prototype query mediation system that demonstrates the utility and effectiveness of these techniques and algorithms. | | | | |
| 14. SUBJECT TERMS Computer Security, Trusted Database Management, Security Policy | | | 15. NUMBER OF PAGES 112 | |
| | | | 16. PRICE CODE | |
| 17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED | 18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED | 19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED | 20. LIMITATION OF ABSTRACT UL | |

Contents

| | | |
|----------|---------------------------------------|-----------|
| 1 | Introduction | 1 |
| 1.1 | Problem Statement | 1 |
| 1.2 | Issues | 2 |
| 1.2.1 | Representational Mismatches | 2 |
| 1.2.2 | Semantic Mismatches | 2 |
| 1.2.3 | Security Policy Mismatches | 3 |
| 1.3 | Requirements | 3 |
| 1.3.1 | Autonomy | 3 |
| 1.3.2 | Automation | 4 |
| 1.3.3 | Efficiency | 4 |
| 1.4 | Summary of Results | 4 |
| 2 | Mediation Approach | 6 |
| 3 | Mediation Architecture | 8 |
| 3.1 | Components | 8 |
| 3.2 | Query Mediation | 10 |
| 4 | Semantic Mediation | 13 |
| 4.1 | Mediation Language | 13 |
| 4.2 | Transformation | 14 |
| 4.3 | Model-Free Optimization | 15 |
| 4.3.1 | Folding Algorithms | 15 |
| 4.3.2 | Query Minimization | 17 |
| 4.3.3 | Most General Queries | 17 |
| 4.4 | Translation | 17 |

| | | |
|----------|---|-----------|
| 5 | Security Mediation | 18 |
| 5.1 | Access Control | 19 |
| 5.1.1 | A Hybrid Approach | 20 |
| 5.2 | Security Translation | 20 |
| 5.2.1 | Translation of MAC Policies | 20 |
| 5.2.2 | Sample Translation | 21 |
| 5.3 | Translation Consistency | 22 |
| 5.3.1 | Integrated Security Policies | 23 |
| 6 | Prototype Mediation System | 24 |
| 6.1 | Overview of Mediation | 25 |
| 6.2 | HTML/CGI Interface | 25 |
| 6.3 | Query Translation | 26 |
| 6.4 | Query Transformation | 26 |
| 6.4.1 | Application/Resource Relationships | 27 |
| 6.4.2 | Resource Usage | 27 |
| 6.4.3 | Access Control | 30 |
| 6.5 | Global Query Plan | 31 |
| 6.5.1 | Generation | 31 |
| 6.5.2 | Optimization | 34 |
| 6.5.2.1 | Projection | 34 |
| 6.5.2.2 | Cross Product Elimination | 34 |
| 6.6 | Local Query Plan | 36 |
| 6.7 | Plan Execution | 37 |
| 6.7.1 | Query Dispatcher | 37 |
| 6.7.2 | Result Processor | 38 |
| 6.8 | Wrappers | 38 |
| 6.9 | Knowledge Base | 39 |
| 6.9.1 | Knowledge Base Editor | 39 |
| 6.9.2 | Security Policy Editor | 39 |
| 6.10 | Adding Databases to a Mediated System | 40 |
| 6.11 | Implementation Summary | 41 |
| 7 | Conclusion | 42 |
| 7.1 | Further Work | 42 |
| 7.1.1 | Short- to Medium-Term Requirements | 43 |
| 7.1.2 | Longer-Term Issues | 44 |

| | | |
|----------|---|------------|
| A | System Demonstration Report | A-1 |
| A.1 | Application Scenario | A-1 |
| A.2 | Integration and Mediation | A-3 |
| A.2.1 | Semantic Relationships | A-5 |
| A.2.2 | Security Relationships | A-5 |
| A.3 | System Demonstration | A-7 |
| A.3.1 | Connecting to the System | A-7 |
| A.3.2 | Logging In | A-7 |
| A.3.3 | Selecting an Information Category | A-7 |
| A.3.4 | A Simple Query | A-8 |
| A.3.5 | The Log File | A-8 |
| A.3.5.1 | Interpreting the Log File | A-8 |
| A.4 | Additional Demonstration Queries | A-9 |
| A.4.1 | Access Control | A-9 |
| A.4.2 | All Sources with Access Control | A-9 |
| A.4.3 | Complex Mediation | A-10 |
| A.4.4 | Other Queries | A-10 |
| A.5 | Query Transcripts | A-15 |
| A.5.1 | Simple Query | A-15 |
| A.5.2 | Access Control – Administrative | A-16 |
| A.5.3 | Access Control – Financial | A-22 |
| A.5.4 | All Sources – Clinical | A-26 |
| A.5.5 | All Sources – Personal | A-30 |
| A.5.6 | Complex Query | A-32 |

Chapter 1

Introduction

SRI International has completed a research program on the development of techniques and tools for efficient, trusted interoperation of autonomous heterogeneous information sources. This program has produced novel techniques for integration of heterogeneous databases into large applications, efficient algorithms for optimization and security of semantic interoperation, and a prototype query mediation system that demonstrates the utility and effectiveness of these techniques and algorithms.

1.1 Problem Statement

Trusted interoperation of heterogeneous information sources is a pressing need in both the military and commercial sectors as organizations attempt to share data from disparate databases while maintaining data security. Each database may be developed independently and maintained to serve the needs of a single organizational unit. Secure exchange of data between such databases can be problematic because of differences in representation and semantics of the data as well as differences in the security policies designed to protect data from unauthorized access. Although translators can be constructed to reformat data from one representation to another, such a translation does not guarantee that the combined, translated data are meaningful. In the interoperation of multilevel secure (MLS) databases with system-high legacy databases, similar problems arise in the representation and semantics of security policies.

1.2 Issues

Technology that adequately addresses the problem of trusted semantic interoperability must address three core issues.

1.2.1 Representational Mismatches

In heterogeneous information sources, the same data can be represented in various incompatible structures. Conversely, the same data structures may be used to represent data with incompatible semantics [18]. In general, there does not exist a universal representation that is perfect for every application [1, 6, 8]. Examples of representational mismatches include

- **Identification.** Patients might be identified by locally assigned patient ID numbers in a clinical database, but by Social Security numbers in a billing database. The nature of the operations supported by the different databases requires that different identifiers be used.
- **Biased view.** The many-to-one relationship between patients and their primary physicians can be represented as a binary predicate, a binary Boolean function, a physician attribute attached to patient objects, or a multivalued patient attribute attached to physician objects. It is infeasible to represent the relationship in all possible structures.

1.2.2 Semantic Mismatches

Semantic differences in the data stored in heterogeneous databases arise naturally from the diverse needs of applications. In addition, relationships between heterogeneous data may be incomplete or uncertain. Examples of semantic mismatches include

- **Scope.** A hospital database includes both inpatients and outpatients, while a clinic database includes only outpatients.
- **Granularity.** A laboratory database tracks individual tests, while an insurance database may track panels of tests.
- **Temporal basis.** Transaction time is used in a clinic database, but admission and discharge times are used in a hospital database.

1.2.3 Security Policy Mismatches

The interoperation of heterogeneous databases is further complicated by the need to protect data from unauthorized access. Examples of security mismatches include

- **Classification.** Data in one database may be classified as CONFIDENTIAL, while the same data in another database might be classified as TOP SECRET.
- **Unit of classification.** One database employs element-level classification, while another uses tuple-level classification (row labels) [14].
- **Semantics.** In one database, classification of attribute CARGO means that unauthorized users should not know which flights carry what cargo, but in another it means that unauthorized users should not know which cargo is shipped to which destination.
- **Policy model.** One database may be multilevel secure (MLS) with mandatory access control, while another database is system high with discretionary access control.

1.3 Requirements

Because of the diverse needs of autonomous organizations, heterogeneity will persist rather than disappear. Providing trusted interoperation of autonomous heterogeneous information sources not only makes it possible to share data in isolated databases reliably, but also increases the confidence of users and the willingness of database owners/administrators to participate in the sharing of data. Effective support for trusted interoperation must satisfy three main requirements.

1.3.1 Autonomy

Database autonomy must be respected and preserved. The autonomy requirement takes two forms:

- **Use.** Database users should not be required to learn new query languages or new schemas to access data from multiple sources. Ideally, the only impact

of trusted interoperation on users of information applications should be a positive one, namely, the availability of additional information.

- **Administration.** Trusted interoperation should not result in an additional administrative burden on the owners of participating database systems.

1.3.2 Automation

Trusted interoperation should be automated. Neither users nor administrators should be required to resolve mismatches manually, even if manual resolution is possible. Security policy mismatches require particular care. Security mismatches may be subtle, and manual resolution, even by trusted security officers, can be unreliable. Mismatches should be identified once, when an information source is integrated into an application ("compile time"), and they should be resolved automatically and in real time when the application is used ("run time").

1.3.3 Efficiency

Automated interoperation should be computationally efficient. Mechanisms to enable trusted interoperation should be as simple as possible, yet powerful enough to enable runtime resolution of mismatches across the wide range of data models and query languages in existing database systems. The use of more powerful and more expensive mechanisms should be limited to the integration phase, where the difficult activity of characterizing relationships and mismatches must be performed.

1.4 Summary of Results

The research described in this report builds on work performed at SRI International on a prior DARPA-sponsored research program, entitled "Semantic Interoperation via Intelligent Mediation", which established the viability of a query mediation approach to semantic interoperation. The current effort significantly extends the previous work in three main areas: (1) optimization techniques for increased efficiency of semantic interoperation, (2) scope of interoperability (data models and query languages), and (3) techniques for *trusted* interoperation. More specifically, this report details the following results in the development of techniques and tools for trusted semantic interoperation:

- Novel and efficient algorithms for automated resolution of semantic mismatches in heterogeneous information sources.
- Optimizations for the process of retrieving maximal information from sources containing data relevant to a user's query.
- Translation techniques for unstructured and semistructured information sources, such as those that are maintained on the World Wide Web (WWW).
- Novel and efficient algorithms for automated (run-time) resolution of mismatches in security policies of trusted database systems.
- Automated support for (compile-time) identification of security policy mismatches.
- A prototype query mediation system that demonstrates the application of the techniques and algorithms.

The main text of this report describes the results in more detail. The demonstration of the final prototype system is documented in the appendix.

Chapter 2

Mediation Approach

The ultimate goal of trusted interoperation of heterogeneous databases is for multiple applications to *share* multiple data sources while maintaining *security* of the data sources. There are two aspects to sharing of data sources:

- Multiple applications sharing the same data source.
- Multiple data sources accessed by the same application.

In addition, there are two aspects to maintaining security of data sources:

- **Security.** Any access or information flow denied within a component database must also be denied under trusted interoperation.
- **Autonomy.** Any access or information flow permitted within a component database must also be permitted under trusted interoperation.

Previous work on semantic interoperation [12, 15] has demonstrated the viability of a *mediated* approach to the sharing of data sources. In a mediated system, applications can employ different languages and schemas, and databases can be managed by different database management systems (DBMSs). A mediator has knowledge about different languages and data models, and the relationships between them. When an application issues a query in one language and data model, the mediator transforms the query into other languages and data models using its knowledge, thus enabling the application to access multiple databases without having to know multiple languages and data models.

In the mediation approach described in this report, the objects of mediation are *queries*. Query mediation is a natural approach for the following reasons:

- Queries form the basic unit of interaction between a user (or application) and a data source.
- Queries have a high degree of logical abstraction, and hence are well-suited to automated interoperation techniques.

With queries as the objects of mediation, expressing relationships between different data sources or between a data source and an application becomes a matter of specifying how one or more queries in one source (or application) can be answered by one or more queries in another source.

The mediation approach applies equally well under *trusted* semantic interoperation. In a trusted mediated application, queries (implicitly) become the objects of access control. Using relationships between the security policy of an application and those of component databases, a mediator is able to determine the queries for which a user is authorized, while component databases retain responsibility for enforcing access control over local data. As discussed later, this approach satisfies both the security and autonomy requirements of trusted interoperation.

Chapter 3

Mediation Architecture

As observed in [11], the sharing of data among multiple sources and applications does not necessarily require the sharing of system components. This is certainly true of the mediated approach, where data sharing is viewed primarily as a problem of semantically meaningful data communication [12, 15]. In a *trusted* environment of data sharing, there is the additional requirement that this data communication respect the security policies of component databases.

3.1 Components

A mediator for semantic interoperation of autonomous heterogeneous databases consists of the following components:

- **Mediation language.** Communication between autonomous heterogeneous databases is carried out in a *mediation* language or *interlingua*. This type of language differs from the representation languages (data models) of participating databases. A representation language captures the knowledge about data for the appropriate abstraction and efficient representation of one class of applications, while a mediation language captures the knowledge about data for meaningful and efficient communication among many classes of applications.
- **Knowledge base.** Semantically meaningful communication between heterogeneous databases is based on the relationships between participating databases. These relationships capture the commonalities and mismatches

in semantics and representations between the databases. They are expressed in the *mediation language* and form a *knowledge base*.

- **Query transformer.** A mediation language alone is not sufficient to ensure meaningful communication among heterogeneous databases, because the databases can contain data that mismatch in semantics and representations. A *query transformer* mediates the communication among databases by resolving potential mismatches. Equipped with the knowledge base of relationships between participating databases, the query transformer accepts queries from one database (or application), determines which databases contain relevant data, generates queries to those databases, and mediates the resulting data back to the originator of the query. Mediation is carried out in the mediation language.
- **Translators.** In most cases the query languages and data models of participating databases will differ from the mediation language. For each different database query language/data model, a *translator* is needed to convert queries and data between the database language and the mediation language.
- **Wrappers.** Participating databases are *wrapped* by interface modules that perform the following functions: forwarding queries from a data source to a mediator, forwarding queries from a mediator to the wrapped DBMS, and receiving answers back from a query sent to a mediator. Thus, the wrapper provides communication facilities between a mediator and a data source. Wrappers are largely invisible to users and require little, if any, modification to the interfaces of participating databases.

A trusted mediation system includes the following additional components:

- **Security policies.** Just as the mediator needs knowledge about the representational and semantic relationships between databases, the mediator also requires knowledge of commonalities and mismatches between security policies of participating databases. In a trusted mediation system, relationships between security policies become additional parts of the knowledge base.
- **Security translators.** To enable participating databases to enforce their respective security policies, *security translators* convert security constraints between the mediator's representation and those of the sources. Each wrapper for a trusted data source employs a security policy translator to map

between its policy representation and that of the mediator. The mediator also employs security policy translation as part of the query transformer. These uses of security policy translation are discussed in more detail in Chapter 6.

Each of the components listed thus far is a *run-time* component of the mediation architecture, used during the process of mediating queries. A complete mediation architecture must also include a *compile-time* component, which is used in the construction and maintenance of a mediated information system:

- **Knowledge base editor.** With the run-time components in place, the most difficult task in the construction of a mediated application is the specification of relationships between the various data sources. A *knowledge base editor* provides tools to aid in the construction of the knowledge base. In a trusted mediation system, the knowledge base editor must include a *security policy editor* for specifying relationships between the security policies of participating databases and identifying potential mismatches between the policies.

Figure 3.1 shows the mediation architecture as reflected in the prototype mediation system. The figure shows three autonomous heterogeneous databases interoperating within a mediated application. The components in this system are described in more detail in Chapter 6, while the application and its demonstration are described in the appendix.

3.2 Query Mediation

Suppose that a user, operating at clearance level C , issues query Q to the mediated application via the “HTML Interface” in Figure 3.1. The mediation of query Q would proceed as follows:

1. Query Q is passed from the HTML Interface into a translator that converts Q , expressed in HTML, into Q' , expressed in the mediation language.
2. From query Q' (with clearance C), the Query Transformer uses the Knowledge Base to determine a mediated query Q'_M . In general, Q'_M may be a set of queries targeted to sources A, B, and C. Each query in Q'_M remains at clearance level C .

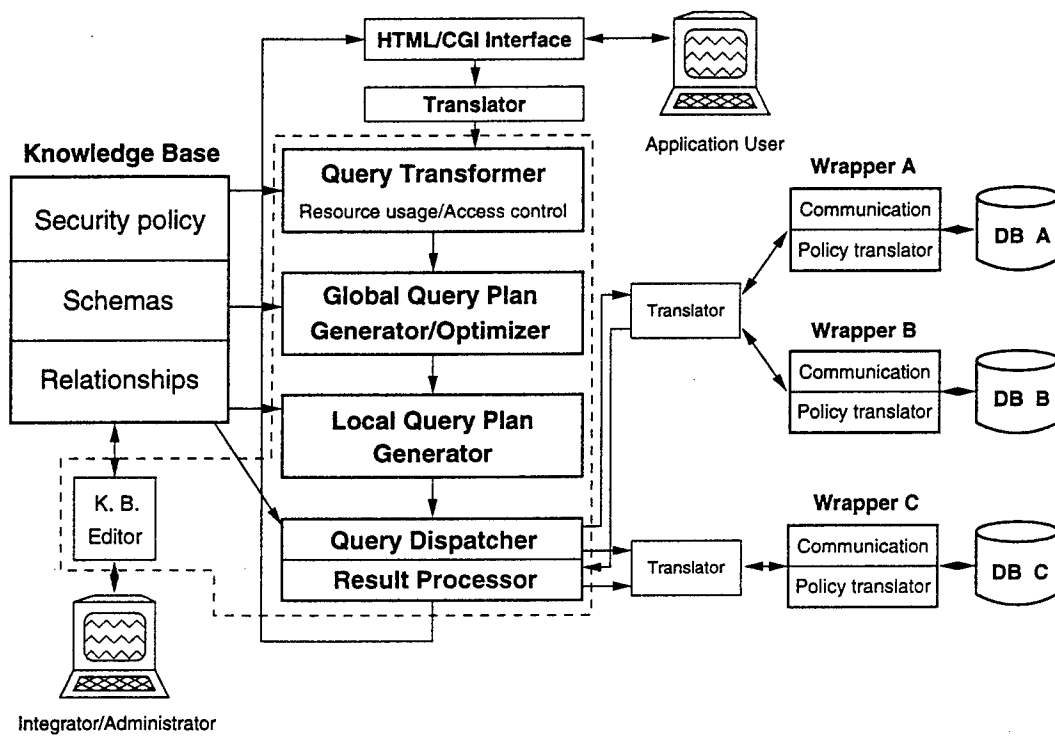


Figure 3.1: Mediation architecture.

3. Mediated query Q'_M is passed to the Global Query Plan Generator, which computes an evaluation plan P for the component queries of Q'_M . Suppose that Q'_M consists of three queries, Q'_A , Q'_B , and Q'_C . Plan P determines where the queries will be sent and in what order.
4. Plan P is passed to the Local Query Plan Generator, which uses knowledge of the sources' query processing capabilities and constraints to convert the queries in P into forms suitable for the sources. The resulting plan P' contains queries Q''_A , Q''_B , and Q''_C .
5. Plan P' is sent to the Query Dispatcher, which sends Q''_A and Q''_B to the common translator for sources A and B, and sends Q''_C to the translator for source C.
6. Queries Q''_A , Q''_B , and Q''_C are translated, respectively, into Q_A , Q_B , and Q_C , which are then forwarded, respectively, to Wrapper A, Wrapper B, and Wrapper C.
7. Wrapper A translates clearance level C into clearance level C_A for source A, forwards Q_A to source A at clearance level C_A , resulting in data D_A , which is passed back to the translator. Wrappers B and C do likewise for Q_B and Q_C , respectively.
8. Answers D_A , D_B , and D_C are translated back into the mediation language, resulting in answers D'_A , D'_B , and D'_C , which are then forwarded to the Result Processor.
9. The Result Processor combines answers D'_A , D'_B , and D'_C , and forwards the result, D , back to the user.

Chapter 4

Semantic Mediation

At a high level, query mediation can be viewed as a three-phase process:

- **Mapping.** The first phase is the compile-time process of determining mappings between
 1. each source-dependent language and a source-independent language (the mediation language), and
 2. queries of one source and those of another source or application.
- **Matching.** The second phase is the run-time process of determining how a user (or application) query can be answered by available sources, resulting in *mediated queries*.
- **Translating.** The third phase is the run-time activity of converting queries from the mediation language into source-specific languages.

4.1 Mediation Language

Central to semantic query mediation is the mediation language. The two principal concerns in the choice of the mediation language are

- **Expressiveness.** The mediation language should be powerful enough to capture the semantics of all data sources participating in a mediated system so that meaningful relationships between the sources can be described.

- **Efficiency.** All the core activities of a mediator are carried out in the mediation language. It is important, therefore, that the language allow for efficient computation.

With these criteria in mind, we have chosen an efficient fragment of first-order logic as the mediation language. This language is essentially an extension of Datalog [20] that preserves both the efficiency and completeness properties of Datalog. Although this language is particularly well suited to mediation among relational systems, it is also suitable for systems employing object query languages [16], such as XSQL [9], and for semistructured information sources.

4.2 Transformation

The key activity in query mediation is query transformation, which determines how a user's query can be answered using available data sources. The result of query transformation is a *mediated query*. There are two important requirements in computing mediated queries:

- **Correctness.** A mediated query should return only information relevant to the user's original query. This implies that there should be *containment* [21] of the mediated query by the original query.
- **Completeness.** A mediated query should return as much relevant information as possible from participating databases.

In a collection of autonomous heterogeneous data sources, two other considerations arise:

- **Partial information.** In general, a single database may not have a complete answer for a user's query. For example, a query might ask for data on all patients, both inpatients and outpatients, but a clinic database may contain data only on outpatients.
- **Multiple sources.** In a heterogeneous environment, many sources may contribute partial information in the answer to a query. For example, one source may contain information on cardiology patients, while another maintains data on oncology patients.

In general, a mediated query is a *collection* of individual queries to sources containing relevant data. Each query in the collection may refer to a single source or to multiple sources. Based on the requirements and considerations for mediated queries, the *answer to a mediated query* is the *union of all answers* (whether partial or complete) from sources containing relevant data.

The transformation of a user/application query into a mediated query is accomplished by a technique known as *query folding*. A description of query folding (with examples) in the context of the prototype is provided in Section 6.4. A complete treatment of query folding is presented elsewhere [13].

4.3 Model-Free Optimization

The query folding technique handles the large and common class of queries known as *project-select-join* (PSJ) queries, also known as *conjunctive* queries. It is well known that the *containment* problem (that is, whether one query yields a subset of the answers of another) for conjunctive queries is computationally intractable (NP-complete) [2]. At present, this result is generally believed to imply that any algorithm for solving the containment problem must be exponential in the sizes of the queries involved. Query folding is a generalization of the containment problem. For a given query and set of resources, query folding computes a set of queries, each of which is guaranteed to be contained in the original query. Hence, query folding must be at least as computationally complex as the containment problem [13]. Nevertheless, there are several ways in which the cost of producing a complete answer to a query over a collection of heterogeneous databases can be reduced.

4.3.1 Folding Algorithms

Various algorithms for query folding, or answering queries using views, have been described in the literature [3, 4, 10, 17, 19]. Each of the algorithms addresses a different aspect of the folding problem, but they have in common the purpose of computing only foldings that are *equivalent* to the original query. Thus, they are not well suited to a heterogeneous environment, where partial information can be useful. Moreover, these algorithms use exhaustive search strategies that are more complex than necessary. To try to regain efficiency, some algorithms prune the search space, sacrificing completeness. Although any complete algorithm for

generating (possibly partial) foldings must be exponential in the worst case, it is possible to do much better than exhaustive search. Furthermore, there is a large class of commonly occurring queries for which folding can be performed in polynomial time. In particular, we have developed the following techniques for optimizing the query transformation process:

- **Query folding for conjunctive (PSJ) queries.** This algorithm generates a complete set of foldings (a mediated query) directly from a user's query and the set of data source relationships from the mediator's knowledge base. Unlike previously proposed algorithms that generate a superset of foldings that must subsequently be filtered, our algorithm is able to detect mismatches as soon as they appear, enabling much more efficient generation of mediated queries. Although the algorithm remains exponential in the worst case, this algorithm can be considered efficient, in that exponential behavior is inherent to the problem, and yet extraneous foldings are not generated.
- **Query folding for acyclic PSJ queries.** This algorithm is a special case of the general algorithm for the commonly occurring class of acyclic queries. The algorithm retains all the advantages of the general algorithm, and has the additional advantage of working in polynomial time for acyclic queries.
- **Query folding for PSJ queries with comparisons.** Comparisons (such as "Age < 55") are among the most commonly used selection conditions in PSJ queries. In the general case, it is believed that the most efficient algorithms for containment of such queries must be doubly exponential [22], holding out little hope for efficient folding of such queries. However, the class of queries with comparisons used most often in practice can be folded with the same efficiency as PSJ queries without comparisons.
- **Query folding for PSJ queries with integrity constraints.** We have developed algorithms for folding of conjunctive queries with functional dependencies and queries with referential integrity constraints. When such integrity constraints are known to hold on participating databases, these folding algorithms may be able to generate foldings that otherwise would have been overlooked. These additional foldings can yield additional information from sources containing relevant data. As with the basic query folding algorithm, the algorithms for folding with integrity constraints are exponential time in the worst case.

4.3.2 Query Minimization

While the query folding algorithms are correct and complete, each individual query in the mediated set is not necessarily in optimal form. Specifically, query folding may generate queries with redundant literals (which represent references to tables in the relational model). Hence, we further optimize the output of query transformation by eliminating redundant references in mediated queries, minimizing their model-independent representation. This minimization results in more efficient source-specific queries at query evaluation time.

4.3.3 Most General Queries

In addition to redundant *literals* within queries, the folding algorithms may also generate redundant *queries*. For a query to be considered redundant, it must be contained in another query in the mediated set. Since redundant queries yield no additional data from participating databases, they should be eliminated. This is done by checking each folding, after it is generated, against the set of foldings already produced. If it is not contained in one of the foldings already produced, it is kept. If it contains one of the foldings already produced, the contained query is discarded. Otherwise, the query itself is contained in one of the others and is discarded. The end result is the set of most general queries. It represents the smallest set of queries that yield complete information from participating databases. An interesting open problem is how to perform query folding so that only most general foldings are generated from the outset.

4.4 Translation

The final phase of semantic mediation is the translation of queries from the mediation language into the appropriate source-specific query language. This process is straightforward for relational languages. For object query languages, the translation process is documented elsewhere [16]. The translation of HTML queries, as required in the prototype mediator, is discussed in Section 6.3.

Chapter 5

Security Mediation

While semantic interoperation techniques are concerned with optimizing mediation and maximizing the amount of relevant information extracted, security mediation techniques focus on ensuring that information is made available only to authorized users. As noted in Chapter 2, there are two guiding principles in the trusted interoperation of autonomous heterogeneous databases:

- **Principle of security.** Any access denied in a component database must also be denied under trusted interoperation. In other words, the security of a database must not be compromised by its participation in an integrated information system.
- **Principle of autonomy.** Any access permitted in a component database must also be permitted under trusted interoperation. In particular, a component database should not have to alter its security policy as a prerequisite to participation in a mediated system.

A solution to trusted interoperation that adheres to both the security and autonomy principles is security mediation. By analogy with semantic interoperation, security mediation works as follows:

1. The security policy of each participating database is mapped to that of a mediated application or another source, and these mappings, or *security relationships*, are maintained in the mediator's knowledge base.
2. Using the security relationships in the knowledge base, the query transformer determines the set of mediated queries for which the issuer of the query has adequate clearance.

5.1 Access Control

A key design decision that must be made in security mediation is the positioning of access control within the mediation architecture (Figure 3.1). We identify three distinct points where access control can be enforced:

- **User interface.** In principle, access control could be enforced solely at the user interface. However, the kind of access control that could be enforced at the interface would be either too weak (checking the user's authorization only at login time) or too strong (authorizing or rejecting queries in their entirety). Hence, we eliminate this possibility from further consideration.
- **Mediator.** Access control is enforced solely at the mediator for all participating data sources.
- **Wrappers.** Access control is enforced by each data source independently.

To evaluate the relative merits of mediator-based access control and wrapper-based access control, we consider the following criteria:

- **Policy management.** How easy or difficult is it to maintain a consistent security policy in the mediated system? How easy or difficult is it to integrate an additional data source into an existing mediated system?
- **Assurance.** How easy or difficult is it to achieve high assurance in a trusted mediation system?
- **Autonomy.** Is the autonomy of participating databases respected?

Enforcing access control at the mediator is advantageous from the standpoint of policy management, presuming that the mediator begins with a consistent view of the security policies of participating sources, since it is relatively straightforward to define policy mappings between a new source and the mediator. On the other hand, autonomy of participating sources is lost. Moreover, high assurance may be difficult or impossible to achieve, since enforcement of access control in the mediator brings the mediator into the trusted computing base (TCB).

Enforcing access control at the wrappers is advantageous from the standpoint of autonomy — sources maintain complete autonomy. It may also be advantageous in terms of assurance, since little mediator functionality must be brought into the TCB. On the other hand, policy management may be more difficult, particularly

if there is no global security policy in the mediated system: consistency must be ensured between all pairs of sources. For the same reason, integration of new sources becomes complicated.

5.1.1 A Hybrid Approach

A reasonable solution to the problem of positioning access control is a hybrid of mediator-based and wrapper-based access control. In the hybrid approach, participating databases maintain and enforce their own local policies, while the mediator maintains and enforces a global policy. This hybrid approach preserves the autonomy of sources and enables relative ease of integration of new sources. Ease of achieving high assurance is unclear, but the hybrid approach is clearly superior to the mediator-based in this respect, while it is similar in assurance to the wrapper-based approach. Based on these findings, we adopt the hybrid approach to access control in our mediation architecture.

5.2 Security Translation

We have developed techniques for security mediation that

1. assist in defining consistent mappings between security policies of data sources, and
2. use these mappings to translate security constraints from the policy of one source or application into the policy of another.

This section describes the translation technique. The technique for defining security policy mappings, the security policy editor, is described in Section 6.9.2. The techniques described in this report focus on mandatory access control (MAC) policies in multilevel secure (MLS) databases, since databases of this type are used in the prototype system. However, these techniques can be readily adapted to other types of policies, such as discretionary access control (DAC) policies in system-high databases.

5.2.1 Translation of MAC Policies

In a secure mediated application, each authorized user logs in at a specific *clearance* level. Each query issued by the user inherits this clearance level when it

is sent to the mediator. The same level is also assigned to each query resulting from query transformation (Section 4.2). When a resulting single-source query is issued to a data source, the source's wrapper translates the clearance level assigned to the query at the mediator into the level appropriate for the source. Thus, while the core components of the mediator are concerned with retrieving the maximum amount of relevant information, security policy translation in each source's wrapper ensures that only information for which the user is cleared is returned.

The portion of a data source's security policy relevant to query mediation is the set of security *levels* used by the source, and the *dominance* relation between them. The security levels and dominance relation together form a security lattice [5]. Relationships between the security policy of a mediated application and the policies of the sources are represented by *cross-lattice* dominance mappings.

Figure 5.1 shows hypothetical cross-lattice dominance relationships between the security lattice of a mediated application and those of two participating sources. Source A uses a simple four-element security lattice, with *low* and *high* levels represented by **Pub** (public) and **Prv** (private), respectively. There are also two (incomparable) intermediate levels, **Pat** (patient) and **Pro** (provider), intended to model the classification of patient-sensitive and provider-sensitive information, respectively. Source B uses a similar lattice, with intermediate levels **Acc** (accounting) and **Ins** (insurers), intended to model the classification of internal accounting information and external insurer information, respectively. In the mediated application, **Acc** dominates **Pat**, while **Ins** dominates both **Pat** and **Pro**. Dashed arrows indicate cross-lattice dominance relationships.

5.2.2 Sample Translation

As an illustration of how the security policy translation is used, consider a query issued in the mediated application at a clearance level of **Acc**. Suppose that there is information relevant to this query in both sources (A and B). There is no direct counterpart to level **Acc** in Source A. However, level **Acc** in the mediator dominates level **Pat**, which does have a direct counterpart (**Pat**) in Source A. Hence, the query to Source A is issued at clearance level **Pat**. On the other hand, Source B does have a direct counterpart to **Acc**, and the query to Source B is issued at level **Acc**.

Now, consider a query issued in the mediator at level **Ins**. Again, Source A has no direct counterpart to **Ins**, but there are two levels in the mediator, **Pat** and

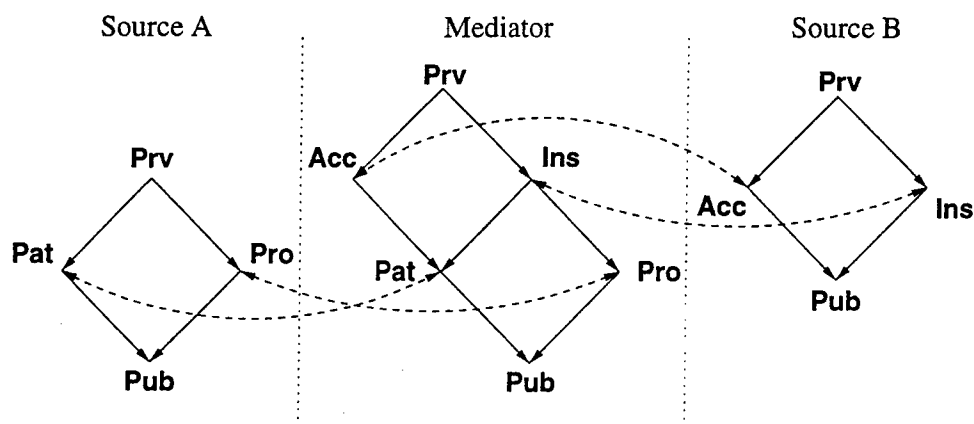


Figure 5.1: Cross-lattice dominance relationships.

Pro, dominated by **Ins**, with corresponding labels in Source A. Hence, a query from the mediator at level **Ins** is issued twice to Source A, once at level **Pat**, and once at level **Pro**. In general, the following translation procedure determines the level(s) at which a query from the mediator to a source must be issued at the source. Let L denote the clearance level of the query at the mediator. Let $\{L'_1, \dots, L'_n\}$, $n > 0$, denote the set of *maximal* (in the dominance order) levels at the *source* that are dominated by L . Then, the set of clearance levels at which the query must be issued at the source is $\{L'_1, \dots, L'_n\}$.

5.3 Translation Consistency

To avoid compromising the security of any source, cross-lattice dominance relationships must be consistent [7]. In viewing the cross-lattice diagram as a directed graph, the dominance relationships are said to be consistent if there are no cycles in the graph involving at least one arc of a component lattice. The relationships represented in Figure 5.1 are consistent. Figure 5.2 represents an *inconsistent* cross-lattice dominance relationship. Level **Pub** in Source B dominates level **Ins** in the mediator, while level **Pro** in the mediator dominates level **Prv** in Source B. As a result, level **Pub** has been effectively equated with level **Prv** in Source B. Consider a mediated environment in which a query is issued from Source B at level **Pub** to the mediator. At the mediator, the query has clearance level **Ins**. The resulting mediated query back to Source B is issued at level **Pro**, which translates

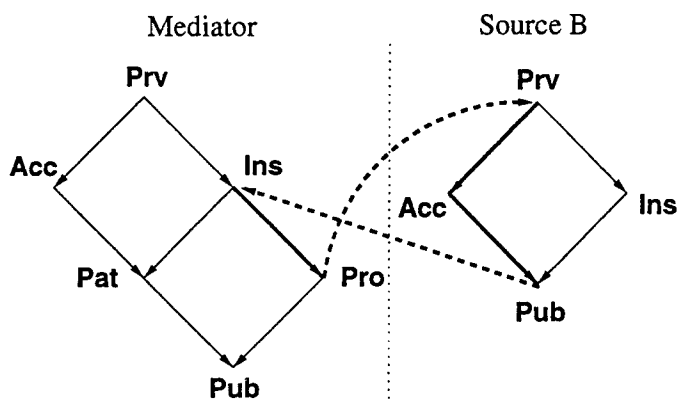


Figure 5.2: Inconsistent dominance relationship.

to **Prv** at Source B, giving the original query (issued at the lowest clearance level) access to data at all levels.

The violation depicted in Figure 5.2 is easily seen. In general, however, security violations resulting from cross-lattice dominance mappings may arise in subtle ways not readily apparent to the administrator/integrator who defines the mappings. For this reason it is important to provide the integrator with automated support for identifying and removing security violations. In the prototype system this functionality is provided in a security policy editor (Section 6.9.2). Using the security policy editor, an integrator effectively resolves potential security policy mismatches at integration time (compile time), enabling automated and efficient policy translation at run time.

5.3.1 Integrated Security Policies

In Figure 5.1, levels in the security policy of the mediated application have close counterparts in the sources, and the cross-lattice dominance relationships indicate equivalence between identically named levels. However, such equivalence is not a requirement for security policy translation. Although in this example the mediator's lattice was derived from those of the sources, this need not be the case in general. All that is required is that a semantically meaningful and consistent mapping be made between the security lattice of the mediator and that of each of the sources.

Chapter 6

Prototype Mediation System

The algorithms and techniques described earlier in this report have been implemented and integrated into a prototype mediation system. The principal components of the mediation architecture are outlined in Chapter 3. Here we describe the components in more detail in the context of a demonstration application¹.

Figure 3.1 in Chapter 3 illustrates the architecture of the prototype system as applied to an information system that integrates three data sources from the health-care and medical research domains. Two of the sources are MLS relational databases containing (actual) data from two units of a health care network. These two units are referred to by the fictitious names “Metropolitan Hospital” and “Valley Clinic”. The third source is the well-known Medline medical research citations database. Medline was developed and is maintained by the National Library of Medicine and is accessible via the World Wide Web. The application as a whole is referred to as MEDINFO. The MEDINFO application can be viewed as a *virtual database*. It has a schema, a query language and data model, and a security policy, but it has *no data* of its own. The data are supplied by information sources participating in the mediated system.

In Figure 3.1, the components within the dashed box are *generic* and constitute the core of the mediator. The wrappers on the right side of the figure are source-specific, although they contain some generic interfacing functionality. The knowledge base and translators can be viewed as components that tie together the core mediator components and the wrapped data sources.

¹The demonstration itself is described in the appendix to this report.

6.1 Overview of Mediation

Before describing the prototype components in more detail, we briefly review the query mediation process. At a high level, the mediation process consists of the following steps:

1. Translate a user's application query from the query language of the application (e.g., HTML) into the mediation language.
2. Transform the application query (now expressed in the mediation language) into (possibly multiple) target database queries (still in mediator language).
3. Remove any target database queries for which the user has insufficient clearance (access control optimization).
4. Generate and optimize a global execution plan for target queries.
5. Generate local query plans for target queries.
6. Execute query plans.
 - (a) Translate target queries from mediator language to database language.
 - (b) Issue target queries on databases. Wrappers at database determine appropriate clearance level for queries by invoking the security policy translator.
 - (c) Perform query processing tasks not handled by databases.
7. Process results and return answers to user query.

The user interface simply provides a means for a user to issue queries in terms of the application schema and to view the answers to that query provided by the mediator. In the demonstration application, the user enters SQL queries into a simple HTML forms interface and views results formatted as HTML tables.

6.2 HTML/CGI Interface

The HTML/CGI interface provides a graphical interface (via a Web browser) to the mediated application. The user logs in at a certain clearance level and then may query the system via fill-out forms. A CGI (Common Gateway Interface) script recasts the HTML query in SQL and forwards it to a translator.

6.3 Query Translation

The first stage in the mediation process is the translation of an application query into the mediation language. In the prototype system, an application query is expressed (after being recast from HTML) in a subset of SQL (restricted to conjunctive, or select-project-join queries). After translation into the mediation language, the user's query is forwarded to the query transformer, which is the primary module of the mediator core.

Translation is also used in a later stage of mediation, during the execution of queries generated by the mediator from an application query. Here the queries must be translated from the mediation language into the query language of the target database. Of the three databases mediated by the demonstration system, two (hospital and medical center) are Trusted Oracle databases, which use SQL as the query language. The third (Medline) uses HTML as its query language. Thus, the prototype system includes modules for translating between the mediation language and SQL and for translating between the mediation language and Medline's HTML encoding. Normally, Medline is accessed via a Web browser by filling out an HTML form. Access to Medline from the mediator is achieved by generating an HTML query that is equivalent to the query generated by the form interface. Thus, Medline remains truly autonomous — its interface need not be modified to enable access from the mediator. Indeed, Medline would be unable to distinguish mediator accesses from Web browser accesses.

6.4 Query Transformation

The query transformer attempts to rewrite a given query, formulated in terms of the application schema, into one or more queries that can be answered by one or more participating databases. As discussed in Section 4.2, the technique to accomplish this rewriting is known as query folding. Using descriptions of the relationships between application and target database schemas, query folding rewrites a query on the application schema into a set of queries on one or more database schemas, if possible. Each query in the resulting set of transformed queries is guaranteed to yield a subset of the answers to the original query. Furthermore, the set of transformed queries produced by query folding is complete, in the sense that the set will yield the maximum amount of information (from the participating databases) relevant to the original query.

6.4.1 Application/Resource Relationships

The relationships between application and resource schemas are described in a general form that relates application queries to resource queries. This means that a relationship description may establish a correspondence between an arbitrary join of relations in the application schema and an arbitrary join of relations in a resource schema. In other words, relationships between application relations and resource relations are many-to-many. The relationship descriptions are created by a person with knowledge of both the application and the database resource with which it is related. Once created, the descriptions are stored in the knowledge base (Section 6.9) for use in the transformation process described in Section 6.4.2.

Figure 6.1 summarizes the relationships between the MEDINFO application schema and the three database sources. The relationships are expressed by rules of the general form

$$application_query \leftrightarrow resource_query$$

where *application_query* and *resource_query* are essentially bodies of Datalog queries formulated in terms of the application and resource schemas, respectively. These rules can be understood as “the query *application_query* in the application schema *corresponds to* the query *resource_query* in the resource schema (for some particular resource)”. Such a rule means that *resource_query* provides *some* answers for *application_query*. In Figure 6.1, the relationships are shown in an abstract form in which the attributes of the relations have been omitted. For example, the first rule in Figure 6.1 states that a query on the relation *Metro.Patients* in the hospital database provides answers for a query on relations *Patient* and *Patient.private* in the application. Examples of fully detailed relationships are given in Figure 6.2, where rules are given that relate patient data from two resources to the application.

6.4.2 Resource Usage

Using the resource descriptions stored in the knowledge base, query folding attempts to rewrite a given application query (expressed in the mediation language) into queries on available database resources. The folding process involves finding a suitable resource replacement for each literal in the body of the application query. If such replacements can be found, and if the conjunction of the replacement literals is consistent with the semantics of the original query, the rewritten

| <u>Application</u> | | <u>Resource</u> |
|---------------------------------------|---|---|
| <i>patient, patient_private</i> | ↔ | <i>metro.patients</i> |
| <i>patient</i> | ↔ | <i>valley.patients</i> |
| <i>physician</i> | ↔ | <i>metro.physicians</i> |
| <i>physician, physician_specialty</i> | ↔ | <i>valley.providers</i> |
| <i>event</i> | ↔ | <i>metro.events, metro.physicians</i> |
| <i>event, visit</i> | ↔ | <i>valley.providers, valley.events</i> |
| <i>research</i> | ↔ | <i>medline.publication, medline.author,</i> <i>medline.keyword</i> |
| <i>research</i> | ↔ | <i>metro.events, metro.physicians</i> |
| <i>research</i> | ↔ | <i>valley.providers, valley.events</i> |

Figure 6.1: Summary of MEDINFO application and resource relationships.

| |
|---|
| <i>patient</i> (<i>Id, Ln, Fn, Adr, Dob, Ms, Sex</i>), <i>patient_private</i> (<i>Id, Ssn, Race</i>) ↔ <i>metro.patients</i> (<i>Id, Ln, Fn, Adr, Ssn, Dob, Ms, Sex, Race</i>) |
| <i>patient</i> (<i>Id, Ln, Fn, Adr, Dob, Ms, Sex</i>) ↔ <i>valley.patients</i> (<i>Id, Ln, Fn, Adr, Dob, Ms, Sex</i>) |

Figure 6.2: Detail of Patient relationships.

query is called a *complete folding*, or, simply, a *folding* of the original query. The folding algorithm used in the prototype mediation system is guaranteed to find all complete foldings of a given application query. This means that the mediator will extract from the participating databases the maximum number of answers for the query. The examples below illustrate the basic concepts underlying query folding.

Consider the following simple query on the (virtual) relation *Patient* in the MEDINFO application:

```
SELECT last_name, first_name FROM Patient.
```

After translation into the mediation language, the query becomes

$$q1(Ln, Fn) :- patient(Id, Ln, Fn, Adr, Dob, Ms, Sex).$$

Examining the resource descriptions in Figure 6.2, we see that there appear to be two resources, *metro.patients* and *valley.patients*, that may serve as replacements for the *patient* literal in the body of *q1*. Indeed, the folding algorithm yields the following two foldings for the query, one for each resource:

$$\begin{aligned} q1a(Ln, Fn) &:- metro.patients(Id, Ln, Fn, Adr, Ssn, Dob, Ms, Sex, Race) \\ q1b(Ln, Fn) &:- valley.patients(Id, Ln, Fn, Adr, Dob, Ms, Sex). \end{aligned}$$

Note that foldings need not have such a simple correspondence between the query and individual resources. Consider the following extension of the query from the example above:

```
SELECT last_name, first_name, race
FROM Patient, Patient_private
WHERE Patient.id = Patient_private.id
```

with translation into the mediation language

$$q2(Ln, Fn, Race) :- patient(Id, Ln, Fn, Adr, Dob, Ms, Sex), \\ patient_private(Id, Ssn, Race).$$

This query requests the attribute *Race* from the logical relation *Patient_private*. Examining the resources in Figure 6.2, we see that this additional information is maintained in the hospital database in relation *metro.patients*, but the medical center database has no corresponding information. Nevertheless, if the two

databases contain some common patient information, the information missing from *valley.patients* might be filled in by *metro.patients*. This possibility is reflected in the queries produced by the folding algorithm:

$q2a(Ln, Fn, Race) :-$
 metro.patients(*Id, Ln, Fn, Adr, Ssn, Dob, Ms, Sex, Race*)
 $q2b(Ln, Fn, Race) :-$
 valley.patients(*Id, Ln, Fn, Adr, Dob, Ms, Sex*),
 metro.patients(*Id0, Ln0, Fn0, Adr0, Ssn0, Dob0, Ms0, Sex0, Race*),
 $Id = Id0.$

In the first folding (query $q2a$) the relation *metro.patients* alone is used to provide answers for $q2$. The second folding (query $q2b$) will attempt to supply answers through a join of *metro.patients* with *valley.patients*. Based on the resource descriptions, queries $q2a$ and $q2b$ are the only foldings that may provide answers to query $q2$. Examples of more complex foldings can be found in the appendix.

6.4.3 Access Control

In a trusted mediated application, the set of queries yielded by query folding generally represents a *superset* of the queries that the user is authorized to issue. Recall from Section 5.1 that the mediator employs a hybrid approach to access control in which the mediator enforces a global policy, while individual sources retain responsibility for access control locally. This implies that, even if the mediator allows the full set of queries to proceed to the sources, the security of the sources will not be violated. Nevertheless, there remain two important reasons for exercising access control in the mediator:

- **Optimization.** If the mediator eliminates, in advance, all mediated queries for which the user has insufficient clearance, mediation becomes more efficient, since queries that would have been rejected at sources do not go through the remaining stages (plan generation, translation, and communication to wrappers) of the mediation process.
- **Systems with no local access control.** Access control in the mediator can also be used as a means of enforcing an access control policy for sources (such as legacy databases) that do not provide local access control. Of

course, for this form of access control to be effective, there must not be alternative means of accessing such sources, which, in turn, implies a loss of autonomy of the sources.

6.5 Global Query Plan

If query transformation is successful, we have a set of queries to be issued to one or more databases. Some of the queries may be answerable completely by one database. For example, queries *q1a*, *q1b*, and *q2a* (from Section 6.4.2) are all single-database queries. Others may involve multiple databases, for example, query *q2b*, which involves both the hospital database and the medical center databases. Individual queries involving multiple databases must be broken down, or decomposed, into subqueries that can be answered by individual databases. Some portions of a query may not be answerable by any database source. Such portions include joins between subqueries on different sources and the evaluation of selection conditions not supported by a source. These parts of the query must then be evaluated by the mediator.

The details of how the set of transformed queries will be evaluated are represented in a *global query plan*. The global plan specifies the order of evaluation of individual queries in the set, how the individual queries are broken down, if necessary, into subqueries, and what processing remains to be done by the mediator.

6.5.1 Generation

A query plan consists of a sequence of subplans. Each subplan is made up of a group of query plans that can (in principle) be executed in parallel. Each query plan in a group initially represents the evaluation of one query from the set of queries produced by the transformation stage. In general, a query plan for any such query may in turn be made up of a sequence of subplans, since, as already mentioned, the query may need to be decomposed into smaller, single-source or mediator-evaluated queries.

In principle, all the queries produced in the transformation stage can be evaluated in parallel, since each query is independent of the others. In practice, the ability to evaluate the queries in parallel depends on the capabilities of the database resources (since several queries may need to be issued to the same database source) and the communication, storage, and processing resources available to the media-

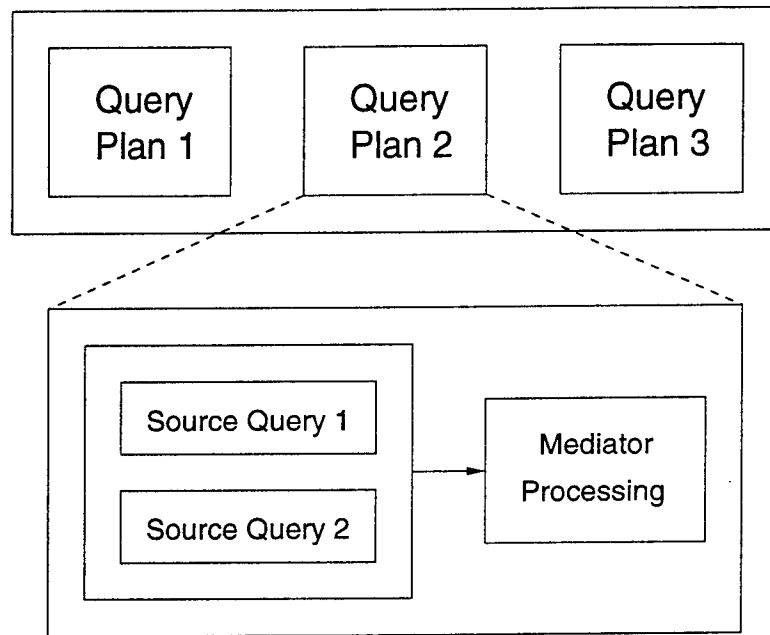


Figure 6.3: Sample global query plan structure.

tor. In the prototype system, the degree of actual parallel evaluation is determined at the plan execution stage (query dispatch and result processing). Hence, the global plan is generated to represent the maximum amount of parallelism possible in principle. This implies that the global plan will consist of a single group of query plans, one for each query in the transformed set.

Figure 6.3 depicts the structure of a sample global query plan for three queries. Each box represents a query plan. Plans grouped together within a box may be evaluated in parallel, while plans connected by an arrow represent a sequence of plan executions. For example, query plan 2 within the global plan consists of a sequence of two subplans. The diagram for the first subplan in the sequence denotes a group of two plans for single-source queries that may be evaluated in parallel. The second subplan in the sequence represents mediator processing that requires the results from the two single-source queries.

Generation of an evaluation plan for each query in the transformed set involves the following steps:

1. Identification of the database source for each literal in the body of the query.

2. Decomposition of the query into single-source queries.
3. Formulation of a query for mediator processing, if necessary.

The following example illustrates these steps. Consider the transformed query $q2b$ from Section 6.4.2:

$q2b(Ln, Fn, Race) :-$
 $valley.patients(Id, Ln, Fn, Adr, Dob, Ms, Sex),$
 $metro.patients(Id0, Ln0, Fn0, Adr0, Ssn0, Dob0, Ms0, Sex0, Race),$
 $Id = Id0.$

Using resource information maintained in the mediator's knowledge base, the plan generator identifies the first literal in the query as a reference to the medical center database, while the second literal refers to the hospital database. The final literal, $Id = Id0$, expresses a join condition between relations from different databases and must be evaluated by the mediator. Thus, the query is decomposed into the following two single-source queries:

$q2b1(Id, Ln, Fn, Adr, Dob, Ms, Sex) :-$
 $valley.patients(Id, Ln, Fn, Adr, Dob, Ms, Sex)$
 $q2b2(Id0, Ln0, Fn0, Adr0, Ssn0, Dob0, Ms0, Sex0, Race) :-$
 $metro.patients(Id0, Ln0, Fn0, Adr0, Ssn0, Dob0, Ms0, Sex0, Race)$

and a mediator-evaluated query to perform the join and return answers to the original query:

$q2b3(Ln, Fn, Race) :-$
 $q2b1(Id, Ln, Fn, Adr, Dob, Ms, Sex),$
 $q2b2(Id0, Ln0, Fn0, Adr0, Ssn0, Dob0, Ms0, Sex0, Race),$
 $Id = Id0.$

Queries $q2b1$ and $q2b2$ may be evaluated in parallel, while the evaluation of $q2b3$ must wait for the evaluation of $q2b1$ and $q2b2$ to complete. Notice that the plan for query $q2b$ thus corresponds to query plan 2 from Figure 6.3.

6.5.2 Optimization

When a mediator system incorporates databases distributed over a wide area (e.g., the Internet), a major factor in the cost of mediation is the amount of network traffic between the mediator and databases. If the global query plan is generated naively, its execution may result in queries on individual databases that yield much more data, and hence result in much more network traffic, than necessary. In the prototype system, optimizations are performed that eliminate two principal sources of unnecessary network traffic, both of which result from the decomposition of multiple-source queries into single-source queries.

6.5.2.1 Projection

The decomposition of query $q2b$ in Section 6.5.1 resulted in two queries, each of which returns an entire table from its respective database. It is clear that not all the information from those relations is needed to answer query $q2b$. In the prototype system a simple optimization is performed on decomposed queries so that only the data for necessary attributes is retrieved. A necessary attribute is one that is part of the projection (occurs in the head of the query) or participates in a join or selection condition. With this optimization the two single-source queries resulting from the decomposition of $q2b$ become

$q2b1(Id, Ln, Fn) :-$

$valley.patients(Id, Ln, Fn, Adr, Dob, Ms, Sex)$

$q2b2(Id0, Race) :-$

$metro.patients(Id0, Ln0, Fn0, Adr0, Ssn0, Dob0, Ms0, Sex0, Race)$

since Ln , Fn , and $Race$ are the projected attributes, while Id and $Id0$ are involved in a join.

6.5.2.2 Cross Product Elimination

Since communication with the database resources constitutes much of the cost of mediation, it might seem that the best decomposition of a multiple-source query is one that minimizes the number of single-source queries produced; that is, after decomposition, there is at most one query for any one database. However, there are often cases in which further decomposition improves execution of the query

plan. For example, consider the following multiple-source query:

```
q3a(Name, Rank) :-  
    metro.physicians(Id, Name, Adr, Lic),  
    metro.role(EvId0, PrId0),  
    metro.events(PatId1, VisId1, EvId1),  
    valley.patients(PatId2, Ln, Fn, Adr, Dob, Ms, Sex),  
    valley.providers(PrId1, Name1, Lic1, Rank),  
    Id = PrId0, EvId0 = EvId1,  
    PatId1 = PatId2, Lic = Lic1
```

which is one of the queries produced by transformation of an application query that requests the names and ranks of all care providers involved in treatment events. Decomposing the query to minimize the number of single-source queries results in the following set:

```
q3a1(Id, Name, Lic, PatId1) :-  
    metro.physicians(Id, Name, Adr, Lic),  
    metro.role(EvId0, PrId0),  
    metro.events(PatId1, VisId1, EvId1),  
    Id = PrId0, EvId0 = EvId1  
  
q3a2(PatId2, Lic1, Rank) :-  
    valley.patients(PatId2, Ln, Fn, Adr, Dob, Ms, Sex),  
    valley.providers(PrId1, Name1, Lic1, Rank)  
  
q3a3(Name, Rank) :-  
    q3a1(Id, Name, Lic, PatId1),  
    q3a2(PatId2, Lic1, Rank),  
    PatId1 = PatId2, Lic = Lic1
```

Notice that query *q3a2* has no join between *valley.patients* and *valley.providers*, and thus represents the cross product of the two relations. However, query *q3a* involves no cross product, because the two relations in question are joined with

relations from another database. After decomposition, these joins are computed by the mediator query $q3a3$. The cross product in $q3a2$ is simply a result of decomposing the multiple-source query $q3a$.

The cross product query suffers from two major drawbacks. First, the target database (in this case, medical center) must compute the cross product, which may involve considerable time and significant processing resources at the database. Second, the cross product (which in general will be quite large) must be communicated back to the mediator, stored, and processed again. Since query $q3a$ involves no cross product to begin with, the overhead of the cross product query can be avoided by decomposing query $q3a2$ and reformulating query $q2a3$, as follows:

$$\begin{aligned} q3a2a(PatId2) &:- valley.patients(PatId2, Ln, Fn, Adr, Dob, Ms, Sex) \\ q3a2b(Lic1, Rank) &:- valley.providers(PrId1, Name1, Lic1, Rank) \\ \\ q3a3(Name, Rank) &:- q3a1(Id, Name, Lic, PatId1), \\ &\quad q3a2a(PatId2), \\ &\quad q3a2b(Lic1, Rank), \\ &\quad PatId1 = PatId2, Lic = Lic1 \end{aligned}$$

Now, query $q2a2$ has been replaced by queries $q2a2a$ and $q2a2b$, which simply retrieve parts of the individual relations referenced in $q2a2$, and $q2a3$ computes the join of these relations with the result of $q2a1$ (which remains unchanged).

6.6 Local Query Plan

In a heterogeneous environment, not all databases will necessarily have the same query evaluation capabilities. For example, some databases may not be able to evaluate certain built-in predicates (e.g., arithmetic comparisons) supported by other databases or by the application. In addition, some databases may have certain constraints on how queries may be formulated; for example, an attribute may be required to have a value supplied in the query (input only), or may not permit a value to be supplied in the query (output only).

In the demonstration system, the Medline text retrieval system has constraints on the forms of queries that may be issued. For example, document searches in Medline must be restricted to particular ranges of dates, such as $1979 < year < 1984$, but no other uses of inequality operators are permitted. In addition, Medline can search for documents based on keyword, but does not return keywords in the

search result. Thus, keyword is used as an input-only attribute. On the other hand, Medline provides descriptive text for documents retrieved in a search, but does not allow search based on such text, so that this descriptive text is effectively an output-only attribute.

To ensure that individual source queries can be evaluated, a local query plan is generated for each such query in the global plan. Each local query plan can be viewed as a refinement of a source query node in the global plan. In the simplest case, the capabilities of the source are sufficient, the query meets the requirements of the source, and hence no refinement is needed. When the query contains (built-in) predicates not evaluable by the source, the query must be decomposed into a query evaluable by the source (if possible) and the remainder that must be processed by the mediator. If the local plan generator determines that the (possibly decomposed) query does not meet the constraints of the source, the mediator can avoid sending the query to the source and instead supply a null answer to the query.

Note that, after local query plan generation, no local query optimization is attempted. Local query plans represent queries entirely answerable by a single database source, and thus optimization of the queries is left to the respective databases.

6.7 Plan Execution

The final stage in the mediation process is the execution of the global query plan and, in turn, the local query plans that constitute the global plan. As the system architecture diagram (Figure 3.1) shows, the plan execution component of the mediator consists of two subcomponents: (1) a query dispatcher and (2) a result processor.

6.7.1 Query Dispatcher

As its name suggests, the query dispatcher is responsible for issuing the individual queries contained in the global query plan to the appropriate database sources or to the mediator's internal query processor. The order of evaluation is specified in the global plan, and the query dispatcher may issue the queries in any manner that satisfies that order. In particular, queries that are grouped together in the plan for parallel evaluation may be evaluated in any arbitrary order, subject to

the constraints of the database sources (for multiple simultaneous connections) and the mediator's own internal resources. In the demonstration prototype, the dispatcher issues grouped queries in an arbitrary sequential order.

Before any database query can be issued, it must be translated from the mediation language into the database's query language. Using information supplied in each local query plan, the dispatcher identifies the appropriate translator and calls it. The dispatcher then sends the translated query to the source specified in the plan. In the case of a mediator query, no translation is required, and it is sent directly to the mediator's internal query processor.

6.7.2 Result Processor

The result processor is responsible for combining the answers returned from database sources and formatting the processed answers for return to the user (the issuer of the original application query). The processing necessary for combining the answers to individual database queries is specified by the query plan in the form of mediator queries. Recall that this processing includes computing joins of relations from different databases as well as the evaluation of built-ins that the query processors of some databases may not handle. Thus, the mediator contains a query processor capable of evaluating select-project-join queries.

For return of the results to the application user, the mediator can be configured in either of two ways. In one configuration, the result processor computes the union of all answers to the application query and returns them as a single set. In the alternate configuration, the mediator returns the answers to each query in the set produced by query transformation (folding) separately, along with an indication of the source(s) of the information.

6.8 Wrappers

Each database incorporated into a mediated system must have a wrapper module with which it communicates with the mediator. The purpose of a wrapper is to accept queries from the mediator, forward these queries to the query processor of the database, accept answers back from the database, and return these results back to the mediator. In a trusted environment, a wrapper must also provide a security policy translator to ensure consistent and meaningful use of security

information in the mediation process. The translation process is described in Section 5.2.

6.9 Knowledge Base

The knowledge base component maintains information regarding schemas and security policies of participating databases and the relationships between the application and each of the participating databases. The query transformer, the query language/data model translators, and the security policy translators all rely on information maintained in the knowledge base. Thus, the knowledge base is central to the mediator. Population of the knowledge base with meaningful and consistent knowledge of the sources and their relationships to the application is critical to the construction of a successful mediated application. In a sense, an instance of the mediation architecture with an empty knowledge base can be viewed as a mediator “template” or “framework”, which upon construction of the knowledge base becomes a mediator.

6.9.1 Knowledge Base Editor

The knowledge base is populated and maintained by an administrator interacting with the knowledge base editor. The main activities supported by a knowledge base editor are

- Description of source schemas, language constraints, and representation structures.
- Specification of relationships between source schemas, in the form of mappings that relate queries answerable by one source to queries answerable by another source or application.

6.9.2 Security Policy Editor

In addition to the knowledge required for semantic mediation, the knowledge base contains information on security policies and their relationships for trusted interoperation. A security policy editor is provided that enables the administrator to perform the following security-related activities:

- Description of security policies of the mediated application and data sources.

- Specification of mappings between the security policy of one source and that of another source or application.
- Identification and resolution of potential security violations that may result from interoperation.

Perhaps the most important function of the security policy editor is the identification of potential security violations. The editor allows the integrator/administrator to specify security relationships one by one. After the addition of each relationship, the editor determines whether the added relationship would permit a security violation. If so, all relationships involved in the potential violation are identified, and the administrator must remove one or more relationships until the violation is corrected. Only then is the addition of new relationships permitted.

6.10 Adding Databases to a Mediated System

In the mediation architecture, incorporating a new database source into a mediated application involves the following activities:

- Creating a wrapper module for the database.
- Creating a translator to translate between the the mediation language and the database language (if such a translator does not already exist).
- Providing descriptions of the schema. of the database, and describing the relationships between the application schema and the database schema.
- Describing the security policy of the database and the relationships between the security policy of the application and that of the database.

Note that, to add a new database, one needs to be familiar only with the application and the database to be added. There is no need for the complex task of schema integration. Nevertheless, the mediation architecture supports a federated-style interoperation approach, where the knowledge base of the mediated application (a virtual federated database) would contain the integrated schema.

6.11 Implementation Summary

The core mediator components, as well as the translators and wrappers, are implemented in ANSI C. The translators are modules that can either be operated as stand-alone programs or be linked directly with the core mediator components. Each wrapper is a stand-alone program that is called as needed by the mediator. The mediator itself runs as a single Unix process, and each database query issued by the mediator is managed by a subprocess of the mediator that calls the appropriate wrapper program. Communication between any wrapper process and its parent process is carried out using POSIX standard pipes to maximize portability of the mediator system. A limited amount of scripting code (e.g., Perl) is used to interface certain stand-alone components and for CGI processing.

The two relational sources are Trusted Oracle databases maintained on a Trusted Solaris platform. The Medline database is maintained separately (by the National Library of Medicine) and made available via the Internet.

Chapter 7

Conclusion

In the research program described in this report, we investigated the problems that arise in trusted semantic interoperability, developed a core set of techniques that address those problems, and implemented and integrated these techniques in a prototype trusted query mediation system. Realistically, this work should be viewed as a significant step toward a general solution to the problem of trusted interoperability of COTS and legacy databases. Although the technology is not yet sufficiently developed to enable ready application of trusted mediation techniques to real systems, it is now at a stage where further development can and should be guided by the needs of real-world information system applications.

7.1 Further Work

We identify here several issues that must be addressed to enable real-world application of trusted mediation technology. These issues are divided into two categories: (1) short- to medium-term needs, which includes pressing needs as well as requirements that could be satisfied with the level of technology currently available, and (2) longer-term goals, where the amount of effort required is large and/or more fundamental research into the technological issues is required. Note that the lists of needs and requirements should not be considered exhaustive.

7.1.1 Short- to Medium-Term Requirements

Translators

We currently have translators for a subset of SQL and for a particular form of HTML queries. The translation between SQL and the mediation should be extended to cover SQL more completely. Many of the required tasks here are straightforward, for example, adding the capability to handle more built-in functions and predicates. Other aspects, such as aggregation, subqueries, and union queries will require considerable research.

For object query languages, the research is already in place, and the development of translators should be relatively straightforward.

For HTML queries, the approach we have taken is fairly general, although our process for generating a translator for an HTML-based application is completely manual. Given the proliferation of Web-based information sources, automating the generation of translators between HTML and the mediation language is worth pursuing.

For other languages and legacy systems, the development of translators can be addressed as required.

Integration Tools

Perhaps the most critical requirement for effective application of trusted mediation technology is the development of graphical tools for automated integration support. As discussed earlier, the successful development of a mediated application depends critically on the population of a knowledge base of schemas, security policies, and relationships.

Knowledge Base Management

Currently, the mediator's knowledge base is maintained as a collection of structured text files. While this is suitable for small applications, it is unlikely to scale well. A more efficient and robust means for storing the knowledge base should be developed, perhaps using available COTS systems. Note that the development of integration tools is closely tied to the design of the knowledge base.

Wrappers

The implementation of a database wrapper is relatively straightforward, and generally they can be produced as needed. Many of the wrappers functions are common across database systems. Thus, the development of a *wrapper generator* would be desirable.

7.1.2 Longer-Term Issues

Mediator Core

In the longer term, and in particular for large-scale mediated applications, work on making the core mediator components more robust and more capable will have to be done. The core functions of the mediator fall broadly into two categories:

1. Query transformation capabilities
2. Query processing capabilities

As the mediation language is a substantial fragment of first-order logic, query transformation is essentially a problem of logical inference in first-order logic. At the same time, the expressiveness of the mediation language requires that the mediator have the full query processing power of a relational database system. Recent advances in the area of deductive database technology make it possible to consider blending query transformation with query processing. With the addition of disk-resident data processing capabilities, it will become possible to implement a mediator core as a deductive database application.

Push/Pull

The techniques we have developed focus exclusively on the problem of *querying* heterogeneous information sources. No attention has been given (in the current project) to the problem of updates in a trusted heterogeneous environment. Progress in this area will require much research into fundamental problems. It should be noted that work in data warehousing is likely to yield useful results in this area.

Bibliography

- [1] R. J. Brachman. The future of knowledge representation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1082–1092, 1990.
- [2] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on the Theory of Computing*, pages 77–90, 1977.
- [3] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 190–200, 1995.
- [4] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Sixth International Conference on Database Theory (ICDT '97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 56–70. Springer-Verlag, 1997.
- [5] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.
- [6] M. L. Ginsberg. Knowledge Interchange Format: The KIF of death. *AI Magazine*, 12(3):57–63, 1991.
- [7] L. Gong and X. Qian. Computational issues in secure interoperation. *IEEE Transactions on Software Engineering*, 22(1):43–52, January 1996.
- [8] W. Kent. Solving domain mismatch and schema mismatch problems with an object-oriented database programming language. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, 1991.

- [9] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 393–402, 1992.
- [10] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems*, pages 95–104, 1995.
- [11] R. Neches, R. E. Fikes, T. Finin, T. Gruber, R. S. Patil, T. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.
- [12] X. Qian. Semantic interoperation via intelligent mediation. In *Proceedings of the Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pages 228–231, April 1993.
- [13] X. Qian. Query folding. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 48–55, 1996.
- [14] X. Qian and T. F. Lunt. Tuple-level vs. element-level classification. In *Database Security VI: Status and Prospects*, pages 301–315. North-Holland, 1993.
- [15] X. Qian and T. F. Lunt. Semantic interoperation: A query mediation approach. Technical Report SRI-CSL-94-02, Computer Science Laboratory, SRI International, April 1994.
- [16] X. Qian and L. Raschid. Query interoperation among object-oriented and relational databases. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 271–278, March 1995.
- [17] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems*, pages 105–112, 1995.
- [18] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In *Proceedings of the IFIP TC2/WG2.6 Conference on Semantics of Interoperable Database Systems*. Elsevier Scientific Publishers, November 1992.

- [19] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proceedings of the Twentieth International Conference on Very Large Data Bases*, pages 367–378, 1994.
- [20] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.
- [21] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1988.
- [22] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proceedings of the Eleventh ACM Symposium on Principles of Database Systems*, pages 331–345, 1992.

Appendix A

System Demonstration Report

This appendix describes the demonstration of trusted mediation technology as applied to an information integration problem in the health-care and medical information domains. We begin by describing the application scenario. We then briefly review the system architecture and mediation process. Finally, we describe the demonstration in detail. A detailed account of the research program that produced the demonstration appears in the main text of this report.

A.1 Application Scenario

The application is a medical information system referred to as "MEDINFO". This application is designed to model a hypothetical integration of separately maintained health-care databases and a Web-based medical citations system into a mediated information system. Specifically, the goal is to create a mediated application that

- Integrates the three sources
- Respects the autonomy of the sources
- Preserves the security of the sources.

While MEDINFO has its own schema and security policy, it has no data of its own, and thus can be viewed as a virtual database. The data for MEDINFO is provided by a collection of three data sources, each with its own schema, semantics, security policy, and query language. Two of the sources are multilevel

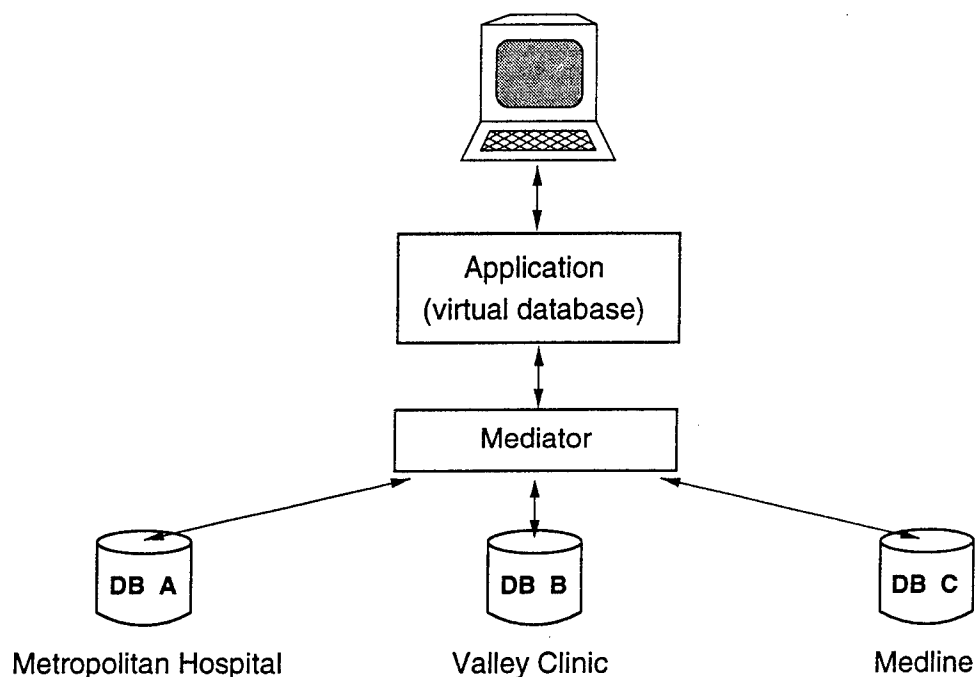


Figure A.1: MEDINFO application.

secure (MLS) relational databases containing actual data from a health-care system, modeled here as a hospital and a clinic. In this report, the hospital and clinic are referred to by the fictitious names "Metropolitan Hospital" and "Valley Clinic", respectively, and their respective databases are referred to as the hospital and clinic databases. Both hospital and clinic databases use Trusted Oracle as the trusted database management system. The third source is the Medline medical research citations system. Unlike the hospital and medical center databases, Medline is a semistructured text source, normally accessed on the World Wide Web (WWW) via a Hypertext Markup Language (HTML) forms interface.

The three sources are linked to the application (virtual database) via a mediator. Figure A.1 is a high-level view of the demonstration system. The user issues a query to the application interface, which then forwards the query to the mediator. The mediator determines whether and how the query may be answered based on its knowledge of the relationships between the application and the participating databases. The mediator then issues queries to each and all of the databases capable of supplying information to answer the application query. Upon receiving the

requested information from the databases, the mediator combines the information appropriately and returns the results to the application interface for presentation to the user.

A.2 Integration and Mediation

Figure A.2 provides a more detailed view of the prototype system. At a high level, the mediation process consists of the following steps:

1. Translate a user's application query from the query language of the application (e.g., HTML) into the mediation language.
2. Transform the application query (now expressed in the mediation language) into (possibly multiple) target database queries (still in mediator language).
3. Remove any target database queries for which the user has insufficient clearance (access control optimization).
4. Generate and optimize a global execution plan for target queries.
5. Generate local query plans for target queries.
6. Execute query plans.
 - (a) Translate target queries from mediator language to database language.
 - (b) Issue target queries on databases. Wrappers at database determine appropriate clearance level for queries by invoking the security policy translator.
 - (c) Perform query processing tasks not handled by databases.
7. Process results and return answers to user query.

The key to constructing a trusted mediated application based on the architecture depicted in Figure A.2 is the specification of semantic and security relationships, which are stored in the knowledge base and used by the query transformer, language translators, and security policy translators for mediating queries.

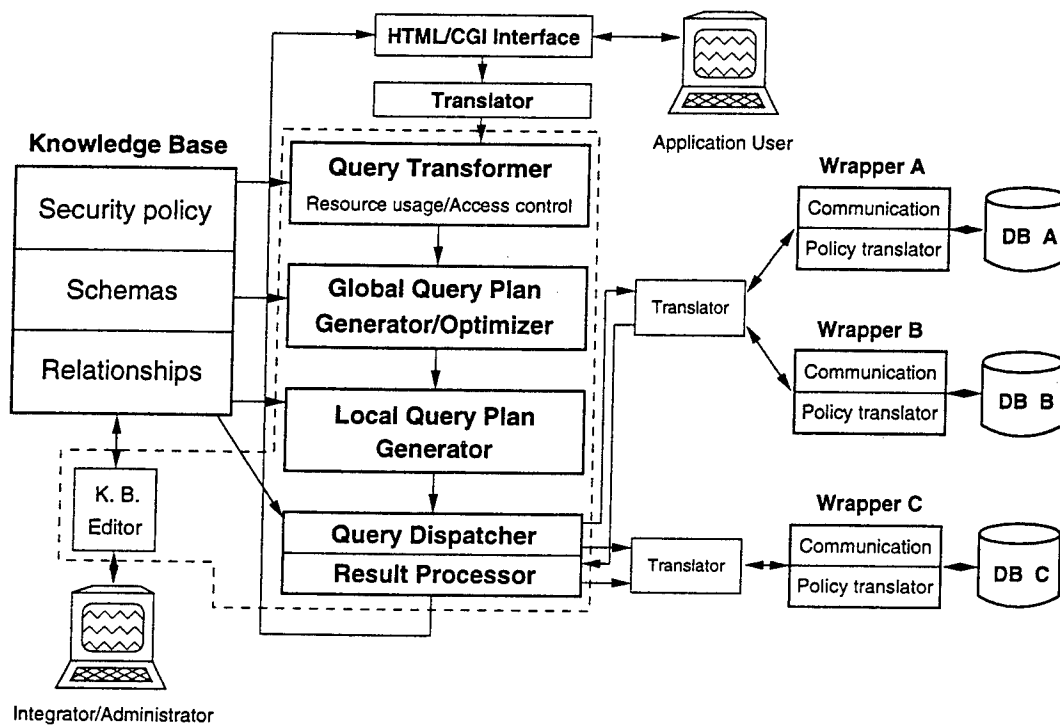


Figure A.2: System architecture.

| <u>Application</u> | | <u>Resource</u> |
|---------------------------------------|---|---|
| <i>patient, patient_private</i> | ↔ | <i>metro.patients</i> |
| <i>patient</i> | ↔ | <i>valley.patients</i> |
| <i>physician</i> | ↔ | <i>metro.physicians</i> |
| <i>physician, physician_specialty</i> | ↔ | <i>valley.providers</i> |
| <i>event</i> | ↔ | <i>metro.events, metro.physicians</i> |
| <i>event, visit</i> | ↔ | <i>valley.providers, valley.events</i> |
| <i>research</i> | ↔ | <i>medline.publication, medline.author, medline.keyword</i> |
| <i>research</i> | ↔ | <i>metro.events, metro.physicians</i> |
| <i>research</i> | ↔ | <i>valley.providers, valley.events</i> |

Figure A.3: Sample semantic relationships.

A.2.1 Semantic Relationships

Semantic relationships are logical rules that establish correspondences between queries answerable by a source and queries answerable by the application. Figure A.3 gives an abstract view of some of the relationships used in the application. For example, the first rule means that a query on the relation *Metro.Patients* in the hospital database can provide answers to a query on the two (virtual) application relations *Patient* and *Patient_private*, while the second rule means that relation *Valley.Patients* in the clinic database can provide answers only to queries on the application relation *Patient*.

A.2.2 Security Relationships

The application and the two relational sources both enforce mandatory access control (MAC) control policies, while Medline is publicly available and enforces no access control policy. Security relationships are modeled as cross-lattice dominance relationships between the security lattices of the application and sources. Figure A.4 depicts these lattices and relationships as used in MEDINFO.

The hospital database (Metro) uses a simple security lattice with three levels: **ADM**, **MED**, and **PER**. Level **ADM** (Administrator) is the highest clearance level at which a user may operate in the hospital database, and is used to protect the most sensitive data in the database. Level **MED** (Medical) is an intermediate

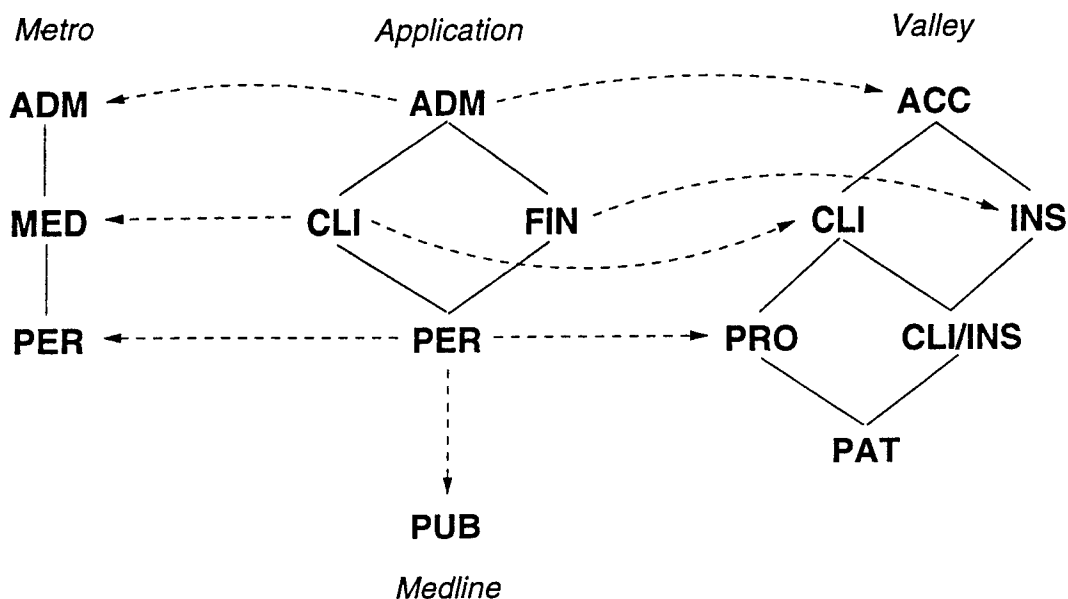


Figure A.4: Security relationships.

clearance/sensitivity level used to label clinical data. The lowest level is **PER** (Personal), used to label personal data pertaining to patients and employees.

The clinic database (Valley) uses a more elaborate security policy. The highest level is **ACC** (Accounting). The next two lower levels (which are incomparable) are **CLI** (Clinical) and **Insurance**, intended to label clinical and insurance-related data, respectively. Level **CLI/INS** reflects the need to label some data for access by both **CLI**- and **INS**-cleared users. Level **PRO** (Provider) labels provider-sensitive data, and the lowest level **PAT** (Patient) labels patient-sensitive data.

The application uses a security policy of intermediate complexity. The top level is **ADM** (Administrative), with lower levels **CLI** (Clinical) and **FIN** (Financial), having intuitive meanings. The lowest level is **PER** (Personal). The dashed arrows in the figure represent cross-lattice dominance relationships. For example, a user operating at clearance level **ADM** in the application has access to all data in all databases, since **ADM** in the application dominates the highest levels in both the hospital and the clinic databases (as well as the sole label **PUB** (Public) in Medline). Similarly, **CLI** in the application dominates **MED** in the hospital database and **CLI** in the clinic database. Level **FIN** in the application dominates **INS** in the clinic database, but has no direct counterpart in the hospital database.

A user operating at clearance level **FIN** may access data labeled at **INS** and below in the clinic database and data labeled at **PER** in the hospital database, since **FIN** dominates **PER** in the application, and **PER** in the application dominates **PER** in the hospital database. Finally, **PER** in the application dominates **PUB** in Medline, which allows users operating at any clearance level full access to Medline.

A.3 System Demonstration

The prototype system is available on the Web to authorized users for demonstration purposes. The system should function well with any reasonably recent graphical Web browser. This section provides guidance on how to use the prototype system to demonstrate its most important aspects. Partial screen dumps that illustrate the demonstration are provided at the end of this appendix.

A.3.1 Connecting to the System

The first step is to connect to the system by pointing a Web browser to the following URL:

`http://www.csl.sri.com/~dawson/mediator/main.html`

which results in display of the system's home page (Figure A.5).

A.3.2 Logging In

The user logs in to the system at one of the four clearance levels shown in Figure A.5: Administrative, Clinical, Financial, or Personal. These levels correspond to the clearance levels for the application depicted in Figure A.4. The system prompts for a username and password to complete the login process. Figure A.6 shows the result of an Administrative-level login.

A.3.3 Selecting an Information Category

For each clearance level there is an associated collection of information categories from which the user can choose. The greatest variety of information is available to an Administrative user (Figure A.6). All other levels have access to some subset of these categories.

A.3.4 A Simple Query

For a simple test of the system, a query for patient information works well. Figure A.7 shows the query form for patient information. The format for other query forms is similar. Information (attributes) to be returned are selected on the left side of the form. Selection conditions (restrictions on the query) are specified by filling in boxes on the right side. Figure A.8 shows the result of selecting the ID number, last name, first name, and date of birth for all patients in the system.

A.3.5 The Log File

While most features of the prototype can be demonstrated simply by querying the system, it is also useful to see what goes on “behind the scene”. The mediator generates a log file that details its activities as it mediates queries. The log file for a query is available upon return of the query results via a hyperlink, labeled “View log file”, that appears at the end of the results.

A.3.5.1 Interpreting the Log File

The log file for the simple patient query (Section A.3.4) appears in Section A.5.1. An entry is made in the log file at each major step in the mediation process:

1. SQL query. The SQL query is the user’s original query generated by the HTML/CGI form interface and formulated in terms of the application schema.
2. Logic (mediator) query. This query is the translation of the user’s query into the mediation language.
3. Foldings. The foldings are the initial result of the query transformation phase. The original query is rewritten into one or more queries to sources containing information relevant to the user’s query.
4. Cleared foldings. The initial set of foldings may include some queries for which the user has insufficient clearance. Although the sources enforce their own local access control policies, the query transformer can optimize the mediation process by removing queries for which the user is not cleared.

5. Query plan. The query plan, as displayed in the log file, is the final output of the query evaluation planning phase. Each node in the plan contains a query that will be issued to one of the three sources (identified by source ID and type) or a query that represents processing that must be performed by the mediator (indicated by type "MEDIATOR").
6. Plan execution. As the plan is executed, queries are dispatched to the appropriate sources. For each query, the result of label translation (performed by the wrappers) is displayed. For example, a query at clearance level "ADM" in the application, when targeted to the hospital database, is issued at level "C ADM1". Similarly, a query targeted to the clinic database is issued at level "C ACC2". Following the clearance level, the type of the source (TOR-ACLE or MEDLINE) is given, along with the translation of the query into the language of the source.

A.4 Additional Demonstration Queries

A.4.1 Access Control

The user logs in at level Administrative¹ and requests information from category Benefit Information. The user wishes to see the names, insurance payors, and policy numbers for all patients in the system, and selects attributes Last name, First name, Payor, and Policy No. The result consists of 575 rows.

Next, the user logs in at level Financial, proceeds to the Benefit Information section, and issues the same query. In this case, only 236 rows are returned, because benefit-related data is labeled at level **ADM** in the hospital database, while corresponding data in the clinic database is labeled at level **INS**. Recall from Section A.2.2 that level **FIN** translates to **PER** in the hospital database. Hence, no information from the hospital database is returned.

A.4.2 All Sources with Access Control

The user logs in at level Clinical or Administrative, and proceeds to category Research Data to find research-related information on hemodialysis. The user

¹The browser's "Back" button can be used to revisit earlier pages in the demonstration. It is not necessary to return to the home page, except when logging in at a different level.

selects all attributes and specifies the keyword “hemodialysis” (keywords are case-insensitive). The result consists of 17 rows. Ten of the rows are medical citations from Medline. Five of the rows are from the hospital database and show the names of physicians and patients involved in treatment events for hemodialysis. Two similar rows are returned from the clinic database.

Next, the user logs in at level Personal, proceeds to the Research Data area, and issues the same query. Now only the 10 rows from Medline are returned, since research information from the hospital and clinic databases pertains to clinical information for which the user has insufficient clearance. The log file shows how the mediator optimizes evaluation of this query by removing the queries that would have been rejected by the hospital and clinic databases.

A.4.3 Complex Mediation

The user logs in at level Clinical or Administrative, and proceeds to the Treatment Event category. The user wishes to see the names, treatment events, and providers of a certain range of patients. The user selects attributes “Last name”, “First name”, “Event description”, and “Provider name”, and specifies a range of 120000 to 125000 for Patient ID². The interesting aspect of this query is the complexity of its mediation. Numerous complex foldings are generated for the query, and the query plan is also quite involved.

A.4.4 Other Queries

The queries discussed in this section demonstrate most of the important features of the mediator, but many other queries are possible, and experimentation with the system is encouraged.

²An unrestricted query can be issued here, but it may take several minutes to evaluate.

Medical Information Mediator

Please select one of the following information areas:

| | |
|-----------------------|------------------|
| <u>Administrative</u> | <u>Financial</u> |
| <u>Clinical</u> | <u>Personal</u> |

Figure A.5: Prototype system home page.

Administrative Area

Information is available in the following categories:

- **Account Summaries**
- **Benefit Information**
- **Patient Charges**
- **Patient Payments**
- **Treatment Events**
- **Patient Personal Data**
- **Provider Personal Data**
- **Research Data**

Figure A.6: Administrative user page.

Patient Information

You may select one or more of the following patient attributes.

For certain attributes you may restrict the information returned to specific values or ranges of values. Note that these attributes need not also be selected. For example, you may request the names and addresses of patients within a certain range of ID numbers.

| | | | | | |
|---|--|--|---|----------------------|--------------------------|
| <input type="checkbox"/> ID | Range: | <input type="text"/> | - | <input type="text"/> | |
| <input type="checkbox"/> Last name | Value: | <input type="text"/> | | | |
| <input type="checkbox"/> First name | Value: | <input type="text"/> | | | |
| <input type="checkbox"/> Street address | | | | | |
| <input type="checkbox"/> City | | | | | |
| <input type="checkbox"/> State | | | | | |
| <input type="checkbox"/> Zip | | | | | |
| <input type="checkbox"/> Date of birth | Range: | <input type="text"/> | - | <input type="text"/> | (date format: DD-MMM-YY) |
| <input type="checkbox"/> Sex | <input checked="" type="checkbox"/> Male | <input checked="" type="checkbox"/> Female | | | |
| <input type="checkbox"/> Marital status | | | | | |
| <input type="checkbox"/> Race | | | | | |
| <input type="button" value="Submit query"/> | | <input type="button" value="Reset form"/> | | | |

Figure A.7: Patient query form.

| pa_pat_id | pa_last_nm | pa_first_nm | pa_dob |
|-----------|------------|-------------|-----------|
| 102431 | REDGRAVE | MICHAEL | 13-JUL-60 |
| 113337 | TAYBACK | VIC | 28-MAR-29 |
| 120990 | CHAPLIN | CHARLIE | 26-JUN-14 |
| 121034 | DORLEAC | FRANCOISE | 05-JUL-31 |
| 121034 | DOUGLAS | DONALD | 05-JUL-31 |
| : | | | |
| 89588 | HAGEN | JEAN | 02-MAY-64 |
| 89625 | MAKEHAM | ELIOT | 19-MAY-54 |
| 89674 | POWELL | WILLIAM | 18-FEB-22 |
| 89709 | WASHBURN | BRYANT | 25-MAR-27 |

Total rows: 300

[View log file](#)

Figure A.8: Patient query results.

A.5 Query Transcripts

A.5.1 Simple Query

SQL query:

```
select pa_pat_id, pa_last_nm, pa_first_nm, pa_dob from patient
```

Logic (mediator) query:

```
q(T1C1,T1C2,T1C3,T1C11) :-  
    med:patient(T1C1,T1C2,T1C3,T1C4,T1C5,T1C6,T1C7,  
                T1C8,T1C9,T1C10,T1C11,T1C12,T1C13)
```

Foldings:

```
q(ID0,LN1,FN2,DOB10) :-  
    med1:patient_table1(ID0,LN1,FN2,MI3,ADR4,CTY5,ST6,  
                        ZIP7,CTR8,SSN9,DOB10,MS11,SEX12,RACE)  
q(ID16,LN17,FN18,DOB26) :- med2:patient_table2(ID16,LN17,FN18,  
        MI19,ADR20,CTY21,ST22,ZIP23,CTR24,SSN25,DOB26,MS27,SEX28)
```

Cleared foldings:

```
q(ID0,LN1,FN2,DOB10) :- med1:patient_table1(ID0,LN1,FN2,MI3,ADR4,  
        CTY5,ST6,ZIP7,CTR8,SSN9,DOB10,MS11,SEX12,RACE)  
q(ID16,LN17,FN18,DOB26) :- med2:patient_table2(ID16,LN17,FN18,  
        MI19,ADR20,CTY21,ST22,ZIP23,CTR24,SSN25,DOB26,MS27,SEX28)
```

Query plan:

Node list id: 1

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: q_1(ID0,LN1,FN2,DOB10) :-

```
    med1:patient_table1(ID0,LN1,FN2,MI3,ADR4,CTY5,ST6,ZIP7,  
                        CTR8,SSN9,DOB10,MS11,SEX12,RACE)
```

Node list id: 2

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: q_2(ID16, LN17, FN18, DOB26) :-

med2:patient_table2(ID16, LN17, FN18, MI19, ADR20, CTY21,
ST22, ZIP23, CTR24, SSN25, DOB26, MS27, SEX28)

Plan execution:

C ADM1

TORACLE

SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm,
t1.pa_dob FROM med1.patient_table1 t1

C ACC2

TORACLE

SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm,
t1.pa_dob FROM med2.patient_table2 t1

A.5.2 Access Control – Administrative

SQL query:

select pa_last_nm, pa_first_nm, payor, policy_no from benefits,
patient where patient_id = pa_pat_id

Logic (mediator) query:

q(T2C2, T2C3, T1C4, T1C5) :-

med:benefits(T1C1, T1C2, T1C3, T1C4, T1C5, T1C6, T1C7, T1C8),

med:patient(T2C1, T2C2, T2C3, T2C4, T2C5, T2C6, T2C7, T2C8, T2C9,
T2C10, T2C11, T2C12, T2C13),

T1C1 = T2C1

Foldings:

```

q(LN157, FN158, PAYOR64, POL65) :-
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                        ST162, ZIP163, CTR164, SSN165, DOB166,
                        MS167, SEX168, RACE),
    med1:benefit_table1(PID61, VID63, AID62, PAYID, PAYOR64, POL65,
                        DAYS66, RATE67, DED68),
    PID61 = ID156
q(LN173, FN174, PAYOR64, POL65) :-
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                        ST178, ZIP179, CTR180, SSN181, DOB182, MS183,
                        SEX184),
    med1:benefit_table1(PID61, VID63, AID62, PAYID, PAYOR64, POL65,
                        DAYS66, RATE67, DED68),
    PID61 = ID172
q(LN157, FN158, PAYOR72, POL73) :-
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                        ST162, ZIP163, CTR164, SSN165, DOB166, MS167,
                        SEX168, RACE),
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                        RATE75, DED76),
    med2:payor_table2(PAYID, PAYOR72),
    PID69 = ID156
q(LN173, FN174, PAYOR72, POL73) :-
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                        ST178, ZIP179, CTR180, SSN181, DOB182,
                        MS183, SEX184),
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                        RATE75, DED76),
    med2:payor_table2(PAYID, PAYOR72),
    PID69 = ID172

```

Cleared foldings:

```

q(LN157, FN158, PAYOR64, POL65) :-
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                        ST162, ZIP163, CTR164, SSN165, DOB166, MS167,
                        SEX168, RACE),

```

```

med1:benefit_table1(PID61,VID63,AID62,PAYID,PAYOR64,POL65,
                     DAYS66,RATE67,DED68),
PID61 = ID156
q(LN173, FN174, PAYOR64, POL65) :-
med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                     ST178, ZIP179, CTR180, SSN181, DOB182,
                     MS183, SEX184),
med1:benefit_table1(PID61, VID63, AID62, PAYID, PAYOR64, POL65,
                     DAYS66, RATE67, DED68),
PID61 = ID172
q(LN157, FN158, PAYOR72, POL73) :-
med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                     ST162, ZIP163, CTR164, SSN165, DOB166, MS167,
                     SEX168, RACE),
med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                     RATE75, DED76),
med2:payor_table2(PAYID, PAYOR72),
PID69 = ID156
q(LN173, FN174, PAYOR72, POL73) :-
med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                     ST178, ZIP179, CTR180, SSN181, DOB182,
                     MS183, SEX184),
med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                     RATE75, DED76),
med2:payor_table2(PAYID, PAYOR72),
PID69 = ID172

```

Query plan:

Node list id: 1

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: q_1(LN157, FN158, PAYOR64, POL65) :-

```

med1:patient_table1(ID156, LN157, FN158, MI159,
                     ADR160, CTY161, ST162, ZIP163,

```

CTR164,SSN165,DOB166,MS167,
SEX168,RACE),
med1:benefit_table1(PID61,VID63,AID62,PAYID,
PAYOR64,POL65,DAYS66,
RATE67,DED68),
PID61 = ID156

Node list id: 2

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq1(ID172,LN173,FN174) :-

med2:patient_table2(ID172,LN173,FN174,MI175,ADR176,
CTY177,ST178,ZIP179,CTR180,
SSN181,DOB182,MS183,SEX184)

Next: 3

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq2(PID61,PAYOR64,POL65) :-

med1:benefit_table1(PID61,VID63,AID62,PAYID,
PAYOR64,POL65,DAYS66,
RATE67,DED68)

Next: 3

Node list id: 3

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_2(LN173,FN174,PAYOR64,POL65) :-

mq1(ID172,LN173,FN174),
mq2(PID61,PAYOR64,POL65),
PID61 = ID172

Node list id: 4

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq3(ID156, LN157, FN158) :-

med1:patient_table1(ID156, LN157, FN158, MI159, ADR160,
CTY161, ST162, ZIP163, CTR164,
SSN165, DOB166, MS167,
SEX168, RACE)

Next: 5

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq4(PID69, POL73, PAYOR72) :-

med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73,
DAYS74, RATE75, DED76),
med2:payor_table2(_V0, PAYOR72),
_V0 = PAYID

Next: 5

Node list id: 5

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_3(LN157, FN158, PAYOR72, POL73) :-

mq3(ID156, LN157, FN158),
mq4(PID69, POL73, PAYOR72),
PID69 = ID156

Node list id: 6

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: q_4(LN173, FN174, PAYOR72, POL73) :-

med2:patient_table2(ID172, LN173, FN174, MI175, ADR176,

```

CTY177,ST178,ZIP179,CTR180,
SSN181,DOB182,MS183,SEX184),
med2:benefit_table2(PID69,VID71,AID70,PAYID,POL73,
                    DAYS74,RATE75,DED76),
med2:payor_table2(_V0,PAYOR72),
_V0 = PAYID, PID69 = ID172

```

Plan execution:

C ADM1

TORACLE

```

SELECT DISTINCT t1.pa_last_nm, t1.pa_first_nm,
                t2.payor_id_desc, t2.policy_number
FROM med1.patient_table1 t1, med1.benefit_table1 t2
WHERE t2.patient_id = t1.pa_pat_id

```

C ACC2

TORACLE

```

SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm
FROM med2.patient_table2 t1

```

C ADM1

TORACLE

```

SELECT DISTINCT t1.patient_id, t1.payor_id_desc, t1.policy_number
FROM med1.benefit_table1 t1

```

MEDIATOR

```

q_2(LN173, FN174, PAYOR64, POL65) :-
    mq1(ID172, LN173, FN174),
    mq2(PID61, PAYOR64, POL65), PID61 = ID172

```

C ADM1

TORACLE

```

SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm
FROM med1.patient_table1 t1

```

```

C ACC2
TORACLE
SELECT DISTINCT t1.patient_id, t1.policy_number, t2.payor_id_desc
FROM med2.benefit_table2 t1, med2.payor_table2 t2
WHERE t2.payor_id = t1.payor_id

```

```

MEDIATOR
q_3(LN157, FN158, PAYOR72, POL73) :-
    mq3(ID156, LN157, FN158),
    mq4(PID69, POL73, PAYOR72), PID69 = ID156

```

```

C ACC2
TORACLE
SELECT DISTINCT t1.pa_last_nm, t1.pa_first_nm,
                t3.payor_id_desc, t2.policy_number
FROM med2.patient_table2 t1, med2.benefit_table2 t2,
     med2.payor_table2 t3
WHERE t3.payor_id = t2.payor_id AND t2.patient_id = t1.pa_pat_id

```

A.5.3 Access Control – Financial

SQL query:

```

select pa_last_nm, pa_first_nm, payor, policy_no from
    benefits, patient where patient_id = pa_pat_id

```

Logic (mediator) query:

```

q(T2C2, T2C3, T1C4, T1C5) :-
    med:benefits(T1C1, T1C2, T1C3, T1C4, T1C5, T1C6, T1C7, T1C8),
    med:patient(T2C1, T2C2, T2C3, T2C4, T2C5, T2C6, T2C7, T2C8, T2C9,
                T2C10, T2C11, T2C12, T2C13),
    T1C1 = T2C1

```

Foldings:

```

q(LN157, FN158, PAYOR64, POL65) :-
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                        ST162, ZIP163, CTR164, SSN165, DOB166, MS167,

```

```

                                SEX168,RACE),
med1:benefit_table1(PID61,VID63,AID62,PAYID,PAYOR64,POL65,
                                DAYS66,RATE67,DED68),
PID61 = ID156
q(LN173, FN174, PAYOR64, POL65) :-
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                        ST178, ZIP179, CTR180, SSN181, DOB182,
                        MS183, SEX184),
    med1:benefit_table1(PID61, VID63, AID62, PAYID, PAYOR64, POL65,
                        DAYS66, RATE67, DED68),
PID61 = ID172
q(LN157, FN158, PAYOR72, POL73) :-
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                        ST162, ZIP163, CTR164, SSN165, DOB166, MS167,
                        SEX168, RACE),
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                        RATE75, DED76),
    med2:payor_table2(PAYID, PAYOR72),
PID69 = ID156
q(LN173, FN174, PAYOR72, POL73) :-
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                        ST178, ZIP179, CTR180, SSN181, DOB182,
                        MS183, SEX184),
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                        RATE75, DED76),
    med2:payor_table2(PAYID, PAYOR72),
PID69 = ID172

Cleared foldings:
q(LN157, FN158, PAYOR72, POL73) :-
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
                        ST162, ZIP163, CTR164, SSN165, DOB166, MS167,
                        SEX168, RACE),
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                        RATE75, DED76),
    med2:payor_table2(PAYID, PAYOR72),
PID69 = ID156

```

q(LN173, FN174, PAYOR72, POL73) :-

```
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
                        ST178, ZIP179, CTR180, SSN181, DOB182, MS183,
                        SEX184),
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73, DAYS74,
                        RATE75, DED76),
    med2:payor_table2(PAYID, PAYOR72),
    PID69 = ID172
```

Query plan:

Node list id: 1

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq1(ID156, LN157, FN158) :-

```
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160,
                        CTY161, ST162, ZIP163, CTR164,
                        SSN165, DOB166, MS167,
                        SEX168, RACE)
```

Next: 2

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq2(PID69, POL73, PAYOR72) :-

```
    med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73,
                        DAYS74, RATE75, DED76),
    med2:payor_table2(_V0, PAYOR72),
    _V0 = PAYID
```

Next: 2

Node list id: 2

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_1(LN157, FN158, PAYOR72, POL73) :-
mq1(ID156, LN157, FN158),
mq2(PID69, POL73, PAYOR72),
PID69 = ID156

Node list id: 3

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: q_2(LN173, FN174, PAYOR72, POL73) :-

med2:patient_table2(ID172, LN173, FN174, MI175, ADR176,
CTY177, ST178, ZIP179, CTR180,
SSN181, DOB182, MS183, SEX184),
med2:benefit_table2(PID69, VID71, AID70, PAYID, POL73,
DAYS74, RATE75, DED76),
med2:payor_table2(_V0, PAYOR72),
_V0 = PAYID, PID69 = ID172

Plan execution:

C PER1

TORACLE

SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm
FROM med1.patient_table1 t1

% FIN translates to INS and PRO in clinic database. Query issued
% at both labels, fails (due to insufficient clearance) at PRO

C INS2

TORACLE

SELECT DISTINCT t1.patient_id, t1.policy_number, t2.payor_id_desc
FROM med2.benefit_table2 t1, med2.payor_table2 t2
WHERE t2.payor_id = t1.payor_id

C PRO2

TORACLE

SELECT DISTINCT t1.patient_id, t1.policy_number, t2.payor_id_desc

```

FROM med2.benefit_table2 t1, med2.payor_table2 t2
WHERE t2.payor_id = t1.payor_id
ORACLE error:  ORA-00942: table or view does not exist

```

MEDIATOR

```

q_1(LN157, FN158, PAYOR72, POL73) :-
    mq1(ID156, LN157, FN158),
    mq2(PID69, POL73, PAYOR72),
    PID69 = ID156

```

C INS2

TORACLE

```

SELECT DISTINCT t1.pa_last_nm, t1.pa_first_nm,
                t3.payor_id_desc, t2.policy_number
FROM med2.patient_table2 t1, med2.benefit_table2 t2,
     med2.payor_table2 t3
WHERE t3.payor_id = t2.payor_id AND t2.patient_id = t1.pa_pat_id

```

C PRO2

TORACLE

```

SELECT DISTINCT t1.pa_last_nm, t1.pa_first_nm,
                t3.payor_id_desc, t2.policy_number
FROM med2.patient_table2 t1, med2.benefit_table2 t2,
     med2.payor_table2 t3
WHERE t3.payor_id = t2.payor_id AND t2.patient_id = t1.pa_pat_id
ORACLE error:  ORA-00942: table or view does not exist

```

A.5.4 All Sources – Clinical

SQL query:

```

select author, subject, location, date from research
    where keyword = 'HEMODIALYSIS'

```

Logic (mediator) query:

```

q(T1C1, T1C3, T1C4, T1C5) :-
    med:research(T1C1, T1C2, T1C3, T1C4, T1C5),
    T1C2 = 'HEMODIALYSIS'

```

Foldings:

```
q(AU129,TL131,JL132,YR133) :-  
    ml:publication(MID,TL131,JL132,YR133,MO,DT,PT,LNG),  
    ml:author(MID,AU129),  
    ml:keyword(MID,KW130), KW130 = 'HEMODIALYSIS'  
q(NM134,PLN136,'Metropolitan Hospital',DATE137) :-  
    med1:event_table1(PID,VID,EID,DESC135,DATE137),  
    med1:patient_table1(PID,PLN136,PFN,PMI,P5,P6,P7,P8,P9,P10,  
        P11,P12,P13,P14),  
    med1:role_table1(PID1,VID1,EID,PRID,RT,RTD),  
    med1:physician_table1(PRID,NM134,ADR,CTY,ZIP,LIC),  
    DESC135 = 'HEMODIALYSIS'  
q(NM138,PLN140,'Valley Clinic',DATE141) :-  
    med2:event_table2(PID,EID,ADT,DDT,SUBJ,PRID,DATE141),  
    med2:patient_table2(PID,PLN140,PFN,PMI,P5,P6,P7,P8,P9,  
        P10,P11,P12,P13),  
    med2:clinical_table2(CCS,SUBJ,DESC139,TYPE),  
    med2:physician_table2(PRID,NM138,ADR,CTY,ZIP,LIC,SC,SR),  
    DESC139 = 'HEMODIALYSIS'
```

Cleared foldings:

```
q(AU129,TL131,JL132,YR133) :-  
    ml:publication(MID,TL131,JL132,YR133,MO,DT,PT,LNG),  
    ml:author(MID,AU129),  
    ml:keyword(MID,KW130), KW130 = 'HEMODIALYSIS'  
q(NM134,PLN136,'Metropolitan Hospital',DATE137) :-  
    med1:event_table1(PID,VID,EID,DESC135,DATE137),  
    med1:patient_table1(PID,PLN136,PFN,PMI,P5,P6,P7,P8,P9,P10,  
        P11,P12,P13,P14),  
    med1:role_table1(PID1,VID1,EID,PRID,RT,RTD),  
    med1:physician_table1(PRID,NM134,ADR,CTY,ZIP,LIC),  
    DESC135 = 'HEMODIALYSIS'  
q(NM138,PLN140,'Valley Clinic',DATE141) :-  
    med2:event_table2(PID,EID,ADT,DDT,SUBJ,PRID,DATE141),  
    med2:patient_table2(PID,PLN140,PFN,PMI,P5,P6,P7,P8,P9,
```

```

P10,P11,P12,P13),
med2:clinical_table2(CCS,SUBJ,DESC139,TYPE),
med2:physician_table2(PRID,NM138,ADR,CTY,ZIP,LIC,SC,SR),
DESC139 = 'HEMODIALYSIS'

```

Query plan:

Node list id: 1

Node

Dep. count: 1

Source id: 2, Type: MEDLINE

Query: q_1(AU129,TL131,JL132,YR133) :-

```

ml:publication(MID,TL131,JL132,YR133,MO,DT,PT,LNG),
ml:author(_V0,AU129),
ml:keyword(_V1,KW130),
_V1 = MID, _V0 = MID, KW130 = 'HEMODIALYSIS'

```

Node list id: 2

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: q_2(NM134,PLN136,'Metropolitan Hospital',DATE137) :-

```

med1:event_table1(PID,VID,EID,DESC135,DATE137),
med1:patient_table1(_V0,PLN136,PFN,PMI,P5,P6,P7,
P8,P9,P10,P11,P12,P13,P14),
med1:role_table1(PID1,VID1,_V1,PRID,RT,RTD),
med1:physician_table1(_V2,NM134,ADR,CTY,ZIP,LIC),
_V2 = PRID, _V1 = EID, _V0 = PID,
DESC135 = 'HEMODIALYSIS'

```

Node list id: 3

Node

Dep. count: 1

Source id: 5, Type: TORACLE


```

WHERE t4.pr_provider_id = t3.provider_id AND
      t3.event_id = t1.ev_event_id AND
      t2.pa_pat_id = t1.ev_pat_id AND
      t1.ev_event_desc = 'HEMODIALYSIS'

```

C CLI2

TORACLE

```

SELECT DISTINCT t4.pr_name, t2.pa_last_nm,
                'Valley Clinic', t1.ev_date
FROM med2.event_table2 t1, med2.patient_table2 t2,
     med2.clinical_table2 t3, med2.physician_table2 t4
WHERE t4.pr_provider_id = t1.physician_id AND
      t3.clinical_code = t1.ev_subjt AND
      t2.pa_pat_id = t1.ev_pat_id AND
      t3.clinical_code_desc = 'HEMODIALYSIS'

```

A.5.5 All Sources – Personal

SQL query:

```

select author, subject, location, date from research
      where keyword = 'HEMODIALYSIS'

```

Logic (mediator) query:

```

q(T1C1,T1C3,T1C4,T1C5) :-
    med:research(T1C1,T1C2,T1C3,T1C4,T1C5),
    T1C2 = 'HEMODIALYSIS'

```

Foldings:

```

q(AU129,TL131,JL132,YR133) :-
    ml:publication(MID,TL131,JL132,YR133,MO,DT,PT,LNG),
    ml:author(MID,AU129),
    ml:keyword(MID,KW130), KW130 = 'HEMODIALYSIS'
q(NM134,PLN136,'Metropolitan Hospital',DATE137) :-
    med1:event_table1(PID,VID,EID,DESC135,DATE137),
    med1:patient_table1(PID,PLN136,PFN,PMI,P5,P6,P7,P8,P9,P10,
                        P11,P12,P13,P14),

```

```

med1:role_table1(PID1,VID1,EID,PRID,RT,RTD),
med1:physician_table1(PRID,NM134,ADR,CTY,ZIP,LIC),
DESC135 = 'HEMODIALYSIS'
q(NM138,PLN140,'Valley Clinic',DATE141) :-
med2:event_table2(PID,EID,ADT,DDT,SUBJ,PRID,DATE141),
med2:patient_table2(PID,PLN140,PFN,PMI,P5,P6,P7,P8,P9,
                    P10,P11,P12,P13),
med2:clinical_table2(CCS,SUBJ,DESC139,TYPE),
med2:physician_table2(PRID,NM138,ADR,CTY,ZIP,LIC,SC,SR),
DESC139 = 'HEMODIALYSIS'

```

Cleared foldings:

```

q(AU129,TL131,JL132,YR133) :-
ml:publication(MID,TL131,JL132,YR133,MO,DT,PT,LNG),
ml:author(MID,AU129),
ml:keyword(MID,KW130), KW130 = 'HEMODIALYSIS'

```

Query plan:

Node list id: 1

Node

Dep. count: 1

Source id: 2, Type: MEDLINE

Query: q_1(AU129,TL131,JL132,YR133) :-

```

ml:publication(MID,TL131,JL132,YR133,MO,DT,PT,LNG),
ml:author(_V0,AU129),
ml:keyword(_V1,KW130),
_V1 = MID, _V0 = MID, KW130 = 'HEMODIALYSIS'

```

Plan execution:

MEDLINE

/usr/tmp/QD_BAAa15197

Contacting host...

Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Reading...
Finished

A.5.6 Complex Query

SQL query:

```
select pa_last_nm, pa_first_nm, description, name
from event, patient, physician
where patient_id >= 120000 and patient_id <= 125000 and
      event.patient_id = pa_pat_id and
      event.provider_license = physician.license
```

Logic (mediator) query:

```
q(T2C2,T2C3,T1C5,T3C2) :-
    med:event(T1C1,T1C2,T1C3,T1C4,T1C5,T1C6),
    med:patient(T2C1,T2C2,T2C3,T2C4,T2C5,T2C6,T2C7,T2C8,
                T2C9,T2C10,T2C11,T2C12,T2C13),
    med:physician(T3C1,T3C2,T3C3,T3C4),
    T1C3 >= 120000, T1C3 <= 125000, T1C3 = T2C1, T1C6 = T3C4
```

Foldings:

```
q(LN157,FN158,DESC44,NM342) :-
    med1:physician_table1(ID341,NM342,ADR343,CTY,ZIP,LIC344),
    med1:patient_table1(ID156,LN157,FN158,MI159,ADR160,CTY161,
                        ST162,ZIP163,CTR164,SSN165,DOB166,MS167,
```

```

        SEX168,RACE),
med1:event_table1(PID42,VID43,EID40,DESC44,DATE41),
med1:role_table1(PID1,VID1,EID40,PRID,RT,RTD),
med1:physician_table1(PRID,NM,ADR,CTY,ZIP,LIC45),
LIC45 = LIC344, PID42 = ID156, PID42 <= 125000,
PID42 >= 120000
q(LN157, FN158, DESC44, NM346) :-
    med2:physician_table2(ID345, NM346, ADR347, CTY, ZIP, LIC348,
        SC, SR),
    med1:patient_table1(ID156, LN157, FN158, MI159, ADR160, CTY161,
        ST162, ZIP163, CTR164, SSN165, DOB166, MS167,
        SEX168, RACE),
    med1:event_table1(PID42, VID43, EID40, DESC44, DATE41),
    med1:role_table1(PID1, VID1, EID40, PRID, RT, RTD),
    med1:physician_table1(PRID, NM, ADR, CTY, ZIP, LIC45),
    LIC45 = LIC348, PID42 = ID156, PID42 <= 125000,
    PID42 >= 120000
q(LN173, FN174, DESC44, NM342) :-
    med1:physician_table1(ID341, NM342, ADR343, CTY, ZIP, LIC344),
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
        ST178, ZIP179, CTR180, SSN181, DOB182,
        MS183, SEX184),
    med1:event_table1(PID42, VID43, EID40, DESC44, DATE41),
    med1:role_table1(PID1, VID1, EID40, PRID, RT, RTD),
    med1:physician_table1(PRID, NM, ADR, CTY, ZIP, LIC45),
    LIC45 = LIC344, PID42 = ID172, PID42 <= 125000,
    PID42 >= 120000
q(LN173, FN174, DESC44, NM346) :-
    med2:physician_table2(ID345, NM346, ADR347, CTY, ZIP, LIC348,
        SC, SR),
    med2:patient_table2(ID172, LN173, FN174, MI175, ADR176, CTY177,
        ST178, ZIP179, CTR180, SSN181, DOB182,
        MS183, SEX184),
    med1:event_table1(PID42, VID43, EID40, DESC44, DATE41),
    med1:role_table1(PID1, VID1, EID40, PRID, RT, RTD),
    med1:physician_table1(PRID, NM, ADR, CTY, ZIP, LIC45),
    LIC45 = LIC348, PID42 = ID172, PID42 <= 125000,

```

```

PID42 >= 120000
q(LN157,FN158,DESC49,NM342) :-
    med1:physician_table1(ID341,NM342,ADR343,CTY,ZIP,LIC344),
    med1:patient_table1(ID156,LN157,FN158,MI159,ADR160,CTY161,
        ST162,ZIP163,CTR164,SSN165,DOB166,MS167,
        SEX168,RACE),
    med2:event_table2(PID48,EID46,ADT,DDT,SUBJ,PRID,EDT47),
    med2:clinical_table2(CCS,SUBJ,DESC49,TYPE),
    med2:physician_table2(PRID,NM,ADR,CTY,ZIP,LIC50,SC,SR),
    LIC50 = LIC344, PID48 = ID156, PID48 <= 125000,
    PID48 >= 120000
q(LN157,FN158,DESC49,NM346) :-
    med2:physician_table2(ID345,NM346,ADR347,CTY,ZIP,LIC348,
        SC,SR),
    med1:patient_table1(ID156,LN157,FN158,MI159,ADR160,CTY161,
        ST162,ZIP163,CTR164,SSN165,DOB166,MS167,
        SEX168,RACE),
    med2:event_table2(PID48,EID46,ADT,DDT,SUBJ,PRID,EDT47),
    med2:clinical_table2(CCS,SUBJ,DESC49,TYPE),
    med2:physician_table2(PRID,NM,ADR,CTY,ZIP,LIC50,SC,SR),
    LIC50 = LIC348, PID48 = ID156, PID48 <= 125000,
    PID48 >= 120000
q(LN173,FN174,DESC49,NM342) :-
    med1:physician_table1(ID341,NM342,ADR343,CTY,ZIP,LIC344),
    med2:patient_table2(ID172,LN173,FN174,MI175,ADR176,CTY177,
        ST178,ZIP179,CTR180,SSN181,DOB182,
        MS183,SEX184),
    med2:event_table2(PID48,EID46,ADT,DDT,SUBJ,PRID,EDT47),
    med2:clinical_table2(CCS,SUBJ,DESC49,TYPE),
    med2:physician_table2(PRID,NM,ADR,CTY,ZIP,LIC50,SC,SR),
    LIC50 = LIC344, PID48 = ID172, PID48 <= 125000,
    PID48 >= 120000
q(LN173,FN174,DESC49,NM346) :-
    med2:physician_table2(ID345,NM346,ADR347,CTY,ZIP,LIC348,
        SC,SR),
    med2:patient_table2(ID172,LN173,FN174,MI175,ADR176,CTY177,
        ST178,ZIP179,CTR180,SSN181,DOB182,

```

```

MS183,SEX184),
med2:event_table2(PID48,EID46,ADT,DDT,SUBJ,PRID,EDT47),
med2:clinical_table2(CCS,SUBJ,DESC49,TYPE),
med2:physician_table2(PRID,NM,ADR,CTY,ZIP,LIC50,SC,SR),
LIC50 = LIC348, PID48 = ID172, PID48 <= 125000,
PID48 >= 120000

```

Cleared foldings:
 % Identical to Foldings

Query plan:

Node list id: 1

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: q_1(LN157, FN158, DESC44, NM342) :-

```

med1:physician_table1(ID341,NM342,ADR343,CTY,ZIP,
LIC344),

```

```

med1:patient_table1(ID156, LN157, FN158, MI159, ADR160,
CTY161, ST162, ZIP163, CTR164,
SSN165, DOB166, MS167,
SEX168, RACE),

```

```

med1:event_table1(PID42, VID43, EID40, DESC44, DATE41),

```

```

med1:role_table1(PID1, VID1, _V0, PRID, RT, RTD),

```

```

med1:physician_table1(_V1, NM, ADR, _V2, _V3, LIC45),

```

```

_V3 = ZIP, _V2 = CTY, _V1 = PRID, _V0 = EID40,

```

```

LIC45 = LIC344, PID42 = ID156,

```

```

PID42 <= 125000, PID42 >= 120000

```

Node list id: 2

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq1(NM346,CTY,ZIP,LIC348) :-
med2:physician_table2(ID345,NM346,ADR347,CTY,ZIP,
LIC348,SC,SR)

Next: 3

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq2(LN157,FN158,DESC44,_V2,_V3,LIC45) :-

med1:patient_table1(ID156,LN157,FN158,MI159,ADR160,
CTY161,ST162,ZIP163,CTR164,
SSN165,DOB166,MS167,
SEX168,RACE),

med1:event_table1(PID42,VID43,EID40,DESC44,DATE41),

med1:role_table1(PID1,VID1,_V0,PRID,RT,RTD),

med1:physician_table1(_V1,NM,ADR,_V2,_V3,LIC45),

_V1 = PRID, _V0 = EID40, PID42 = ID156,

PID42 <= 125000, PID42 >= 120000

Next: 3

Node list id: 3

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_2(LN157,FN158,DESC44,NM346) :-

mq1(NM346,CTY,ZIP,LIC348),

mq2(LN157,FN158,DESC44,_V2,_V3,LIC45),

_V3 = ZIP, _V2 = CTY, LIC45 = LIC348

Node list id: 4

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq3(NM342,PID42,DESC44) :-

med1:physician_table1(ID341,NM342,ADR343,CTY,ZIP,
LIC344),

med1:event_table1(PID42,VID43,EID40,DESC44,DATE41),
med1:role_table1(PID1,VID1,_V0,PRID,RT,RTD),
med1:physician_table1(_V1,NM,ADR,_V2,_V3,LIC45),
_V3 = ZIP, _V2 = CTY, _V1 = PRID, _V0 = EID40,
LIC45 = LIC344, PID42 <= 125000, PID42 >= 120000

Next: 5

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq4(ID172,LN173,FN174) :-

med2:patient_table2(ID172,LN173,FN174,MI175,ADR176,
CTY177,ST178,ZIP179,CTR180,
SSN181,DOB182,MS183,SEX184)

Next: 5

Node list id: 5

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_3(LN173,FN174,DESC44,NM342) :-

mq3(NM342,PID42,DESC44),
mq4(ID172,LN173,FN174), PID42 = ID172

Node list id: 6

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq5(ID172,LN173,FN174) :-

med2:patient_table2(ID172,LN173,FN174,MI175,ADR176,
CTY177,ST178,ZIP179,CTR180,
SSN181,DOB182,MS183,SEX184)

Next: 7

Node

Dep. count: 1

Source id: 4, Type: TORACLE
Query: mq6(PID42,DESC44,_V2,_V3,LIC45) :-
 med1:event_table1(PID42,VID43,EID40,DESC44,DATE41),
 med1:role_table1(PID1,VID1,_V0,PRID,RT,RTD),
 med1:physician_table1(_V1,NM,ADR,_V2,_V3,LIC45),
 _V1 = PRID, _V0 = EID40, PID42 <= 125000,
 PID42 >= 120000

Next: 7

Node

Dep. count: 1
Source id: 5, Type: TORACLE
Query: mq8(NM346,CTY,ZIP,LIC348) :-
 med2:physician_table2(ID345,NM346,ADR347,CTY,ZIP,
 LIC348,SC,SR)

Next: 8

Node list id: 7

Node

Dep. count: 2
Source id: 0, Type: MEDIATOR
Query: mq7(LN173,FN174,DESC44,_V2,_V3,LIC45) :-
 mq5(ID172,LN173,FN174),
 mq6(PID42,DESC44,_V2,_V3,LIC45), PID42 = ID172

Next: 8

Node list id: 8

Node

Dep. count: 2
Source id: 0, Type: MEDIATOR
Query: q_4(LN173,FN174,DESC44,NM346) :-
 mq7(LN173,FN174,DESC44,_V2,_V3,LIC45),
 mq8(NM346,CTY,ZIP,LIC348),
 _V3 = ZIP, _V2 = CTY, LIC45 = LIC348

Node list id: 9

Node

Dep. count: 1
Source id: 4, Type: TORACLE
Query: mq9(ID156, LN157, FN158) :-
 med1:patient_table1(ID156, LN157, FN158, MI159, ADR160,
 CTY161, ST162, ZIP163, CTR164,
 SSN165, DOB166, MS167,
 SEX168, RACE)

Next: 10

Node

Dep. count: 1
Source id: 5, Type: TORACLE
Query: mq10(PID48, DESC49, _V2, _V3, LIC50) :-
 med2:event_table2(PID48, EID46, ADT, DDT, SUBJ,
 PRID, EDT47),
 med2:clinical_table2(CCS, _V0, DESC49, TYPE),
 med2:physician_table2(_V1, NM, ADR, _V2, _V3, LIC50,
 SC, SR),
 _V1 = PRID, _V0 = SUBJ, PID48 <= 125000,
 PID48 >= 120000

Next: 10

Node

Dep. count: 1
Source id: 4, Type: TORACLE
Query: mq12(NM342, CTY, ZIP, LIC344) :-
 med1:physician_table1(ID341, NM342, ADR343, CTY, ZIP,
 LIC344)

Next: 11

Node list id: 10

Node

Dep. count: 2
Source id: 0, Type: MEDIATOR
Query: mq11(LN157, FN158, DESC49, _V2, _V3, LIC50) :-
 mq9(ID156, LN157, FN158),
 mq10(PID48, DESC49, _V2, _V3, LIC50), PID48 = ID156

Next: 11

Node list id: 11

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_5(LN157, FN158, DESC49, NM342) :-

mq11(LN157, FN158, DESC49, _V2, _V3, LIC50),

mq12(NM342, CTY, ZIP, LIC344),

_V3 = ZIP, _V2 = CTY, LIC50 = LIC344

Node list id: 12

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq13(NM346, PID48, DESC49) :-

med2:physician_table2(ID345, NM346, ADR347, CTY, ZIP,
LIC348, SC, SR),

med2:event_table2(PID48, EID46, ADT, DDT, SUBJ,
PRID, EDT47),

med2:clinical_table2(CCS, _V0, DESC49, TYPE),

med2:physician_table2(_V1, NM, ADR, _V2, _V3, LIC50,
_V4, _V5),

_V5 = SR, _V4 = SC, _V3 = ZIP, _V2 = CTY,

_V1 = PRID, _V0 = SUBJ, LIC50 = LIC348,

PID48 <= 125000, PID48 >= 120000

Next: 13

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq14(ID156, LN157, FN158) :-

med1:patient_table1(ID156, LN157, FN158, MI159, ADR160,

CTY161, ST162, ZIP163, CTR164,

SSN165, DOB166, MS167,

SEX168, RACE)

Next: 13

Node list id: 13

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_6(LN157, FN158, DESC49, NM346) :-

mq13(NM346, PID48, DESC49),

mq14(ID156, LN157, FN158), PID48 = ID156

Node list id: 14

Node

Dep. count: 1

Source id: 4, Type: TORACLE

Query: mq15(NM342, CTY, ZIP, LIC344) :-

med1:physician_table1(ID341, NM342, ADR343, CTY, ZIP,
LIC344)

Next: 15

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: mq16(LN173, FN174, DESC49, _V2, _V3, LIC50) :-

med2:patient_table2(ID172, LN173, FN174, MI175, ADR176,
CTY177, ST178, ZIP179, CTR180,
SSN181, DOB182, MS183, SEX184),

med2:event_table2(PID48, EID46, ADT, DDT, SUBJ,
PRID, EDT47),

med2:clinical_table2(CCS, _V0, DESC49, TYPE),

med2:physician_table2(_V1, NM, ADR, _V2, _V3, LIC50,
SC, SR),

_V1 = PRID, _V0 = SUBJ, PID48 = ID172,

PID48 <= 125000, PID48 >= 120000

Next: 15

Node list id: 15

Node

Dep. count: 2

Source id: 0, Type: MEDIATOR

Query: q_7(LN173, FN174, DESC49, NM342) :-

mq15(NM342, CTY, ZIP, LIC344),
mq16(LN173, FN174, DESC49, _V2, _V3, LIC50),
_V3 = ZIP, _V2 = CTY, LIC50 = LIC344

Node list id: 16

Node

Dep. count: 1

Source id: 5, Type: TORACLE

Query: q_8(LN173, FN174, DESC49, NM346) :-

med2:physician_table2(ID345, NM346, ADR347, CTY, ZIP,
LIC348, SC, SR),
med2:patient_table2(ID172, LN173, FN174, MI175, ADR176,
CTY177, ST178, ZIP179, CTR180,
SSN181, DOB182, MS183, SEX184),
med2:event_table2(PID48, EID46, ADT, DDT, SUBJ,
PRID, EDT47),
med2:clinical_table2(CCS, _V0, DESC49, TYPE),
med2:physician_table2(_V1, NM, ADR, _V2, _V3, LIC50,
_V4, _V5),
_V5 = SR, _V4 = SC, _V3 = ZIP, _V2 = CTY,
_V1 = PRID, _V0 = SUBJ, LIC50 = LIC348,
PID48 = ID172, PID48 <= 125000, PID48 >= 120000

Plan execution:

C MED1

TORACLE

SELECT DISTINCT t2.pa_last_nm, t2.pa_first_nm, t3.ev_event_desc,
t1.pr_name
FROM med1.physician_table1 t1, med1.patient_table1 t2,
med1.event_table1 t3, med1.role_table1 t4,
med1.physician_table1 t5
WHERE t5.pr_zip = t1.pr_zip AND t5.pr_city = t1.pr_city AND

```

t5.pr_provider_id = t4.provider_id AND
t4.event_id = t3.ev_event_id AND
t5.pr_state_license = t1.pr_state_license AND
t3.ev_pat_id = t2.pa_pat_id AND t3.ev_pat_id <= 125000 AND
t3.ev_pat_id >= 120000

```

C CLI2

TORACLE

```

SELECT DISTINCT t1.pr_name, t1.pr_city, t1.pr_zip,
                t1.pr_state_license
FROM med2.physician_table2 t1

```

C MED1

TORACLE

```

SELECT DISTINCT t1.pa_last_nm, t1.pa_first_nm, t2.ev_event_desc,
                t4.pr_city, t4.pr_zip, t4.pr_state_license
FROM med1.patient_table1 t1, med1.event_table1 t2,
     med1.role_table1 t3, med1.physician_table1 t4
WHERE t4.pr_provider_id = t3.provider_id AND
      t3.event_id = t2.ev_event_id AND
      t2.ev_pat_id = t1.pa_pat_id AND t2.ev_pat_id <= 125000 AND
      t2.ev_pat_id >= 120000

```

MEDIATOR

```

q_2(LN157, FN158, DESC44, NM346) :-
    mq1(NM346, CTY, ZIP, LIC348),
    mq2(LN157, FN158, DESC44, _V2, _V3, LIC45),
    _V3 = ZIP, _V2 = CTY, LIC45 = LIC348

```

C MED1

TORACLE

```

SELECT DISTINCT t1.pr_name, t2.ev_pat_id, t2.ev_event_desc
FROM med1.physician_table1 t1, med1.event_table1 t2,
     med1.role_table1 t3, med1.physician_table1 t4
WHERE t4.pr_zip = t1.pr_zip AND t4.pr_city = t1.pr_city AND
      t4.pr_provider_id = t3.provider_id AND
      t3.event_id = t2.ev_event_id AND

```

```
t4.pr_state_license = t1.pr_state_license AND  
t2.ev_pat_id <= 125000 AND t2.ev_pat_id >= 120000
```

C CLI2

TORACLE

```
SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm  
FROM med2.patient_table2 t1
```

MEDIATOR

```
q_3(LN173,FN174,DESC44,NM342) :-  
    mq3(NM342,PID42,DESC44),  
    mq4(ID172,LN173,FN174), PID42 = ID172
```

C CLI2

TORACLE

```
SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm  
FROM med2.patient_table2 t1
```

C MED1

TORACLE

```
SELECT DISTINCT t1.ev_pat_id, t1.ev_event_desc, t3.pr_city,  
                t3.pr_zip, t3.pr_state_license  
FROM med1.event_table1 t1, med1.role_table1 t2,  
    med1.physician_table1 t3  
WHERE t3.pr_provider_id = t2.provider_id AND  
    t2.event_id = t1.ev_event_id AND  
    t1.ev_pat_id <= 125000 AND t1.ev_pat_id >= 120000
```

MEDIATOR

```
mq7(LN173,FN174,DESC44,_V2,_V3,LIC45) :-  
    mq5(ID172,LN173,FN174),  
    mq6(PID42,DESC44,_V2,_V3,LIC45), PID42 = ID172
```

C CLI2

TORACLE

```
SELECT DISTINCT t1.pr_name, t1.pr_city, t1.pr_zip,  
                t1.pr_state_license
```

FROM med2.physician_table2 t1

MEDIATOR

q_4(LN173, FN174, DESC44, NM346) :-
 mq7(LN173, FN174, DESC44, _V2, _V3, LIC45),
 mq8(NM346, CTY, ZIP, LIC348),
 _V3 = ZIP, _V2 = CTY, LIC45 = LIC348

C MED1

TORACLE

SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm
FROM med1.patient_table1 t1

C CLI2

TORACLE

SELECT DISTINCT t1.ev_pat_id, t2.clinical_code_desc, t3.pr_city,
 t3.pr_zip, t3.pr_state_license
FROM med2.event_table2 t1, med2.clinical_table2 t2,
 med2.physician_table2 t3
WHERE t3.pr_provider_id = t1.physician_id AND
 t2.clinical_code = t1.ev_subjt AND t1.ev_pat_id <= 125000 AND
 t1.ev_pat_id >= 120000

MEDIATOR

mq11(LN157, FN158, DESC49, _V2, _V3, LIC50) :-
 mq9(ID156, LN157, FN158),
 mq10(PID48, DESC49, _V2, _V3, LIC50), PID48 = ID156

C MED1

TORACLE

SELECT DISTINCT t1.pr_name, t1.pr_city, t1.pr_zip,
 t1.pr_state_license
FROM med1.physician_table1 t1

MEDIATOR

q_5(LN157, FN158, DESC49, NM342) :-
 mq11(LN157, FN158, DESC49, _V2, _V3, LIC50),

```
mq12(NM342,CTY,ZIP,LIC344),  
_V3 = ZIP, _V2 = CTY, LIC50 = LIC344
```

C CLI2

TORACLE

```
SELECT DISTINCT t1.pr_name, t2.ev_pat_id, t3.clinical_code_desc  
FROM med2.physician_table2 t1, med2.event_table2 t2,  
     med2.clinical_table2 t3, med2.physician_table2 t4  
WHERE t4.ps_spec_rank = t1.ps_spec_rank AND  
      t4.ps_spec_code = t1.ps_spec_code AND  
      t4.pr_zip = t1.pr_zip AND t4.pr_city = t1.pr_city AND  
      t4.pr_provider_id = t2.physician_id AND  
      t3.clinical_code = t2.ev_subjt AND  
      t4.pr_state_license = t1.pr_state_license AND  
      t2.ev_pat_id <= 125000 AND t2.ev_pat_id >= 120000
```

C MED1

TORACLE

```
SELECT DISTINCT t1.pa_pat_id, t1.pa_last_nm, t1.pa_first_nm  
FROM med1.patient_table1 t1
```

MEDIATOR

```
q_6(LN157,FN158,DESC49,NM346) :-  
    mq13(NM346,PID48,DESC49), mq14(ID156,LN157,FN158),  
    PID48 = ID156
```

C MED1

TORACLE

```
SELECT DISTINCT t1.pr_name, t1.pr_city, t1.pr_zip,  
                t1.pr_state_license  
FROM med1.physician_table1 t1
```

C CLI2

TORACLE

```
SELECT DISTINCT t1.pa_last_nm, t1.pa_first_nm,  
                t3.clinical_code_desc, t4.pr_city, t4.pr_zip,  
                t4.pr_state_license
```

```

FROM med2.patient_table2 t1, med2.event_table2 t2,
     med2.clinical_table2 t3, med2.physician_table2 t4
WHERE t4.pr_provider_id = t2.physician_id AND
     t3.clinical_code = t2.ev_subjt AND
     t2.ev_pat_id = t1.pa_pat_id AND
     t2.ev_pat_id <= 125000 AND t2.ev_pat_id >= 120000

```

MEDIATOR

```

q_7(LN173, FN174, DESC49, NM342) :-
    mq15(NM342, CTY, ZIP, LIC344),
    mq16(LN173, FN174, DESC49, _V2, _V3, LIC50),
    _V3 = ZIP, _V2 = CTY, LIC50 = LIC344

```

C CLI2

TORACLE

```

SELECT DISTINCT t2.pa_last_nm, t2.pa_first_nm,
                t4.clinical_code_desc, t1.pr_name
FROM med2.physician_table2 t1, med2.patient_table2 t2,
     med2.event_table2 t3, med2.clinical_table2 t4,
     med2.physician_table2 t5
WHERE t5.ps_spec_rank = t1.ps_spec_rank AND
     t5.ps_spec_code = t1.ps_spec_code AND
     t5.pr_zip = t1.pr_zip AND t5.pr_city = t1.pr_city AND
     t5.pr_provider_id = t3.physician_id AND
     t4.clinical_code = t3.ev_subjt AND
     t5.pr_state_license = t1.pr_state_license AND
     t3.ev_pat_id = t2.pa_pat_id AND t3.ev_pat_id <= 125000 AND
     t3.ev_pat_id >= 120000

```

DISTRIBUTION LIST

| addresses | number of copies |
|---|---------------------|
| MARY DENZ AFRL/IFGB 525 BROOKS RD. ROME, NY 13441-4505 | 5 |
| SRI INTERNATIONAL ATTN: STEVEN DAWSON 333 RAVENSWOOD AVE. MENLO PARK, VA 94025-3493 | 5 |
| AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514 | 1 |
| ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218 | 2 |
| ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714 | 1 |
| RELIABILITY ANALYSIS CENTER 201 MILL ST. ROME NY 13440-8200 | 1 |
| ATTN: GWEN NGUYEN GIDEP P.O. BOX 8000 CORONA CA 91718-8000 | 1 |

| | |
|---|---|
| AFIT ACADEMIC LIBRARY/LDEE 2950 P STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765 | 1 |
| WRIGHT LABORATORY/MTM, BLDG 653 2977 P STREET, STE 6 WRIGHT-PATTERSON AFB OH 45433-7739 | 1 |
| ATTN: GILBERT G. KUPERMAN AL/CFHI, BLDG. 248 2255 H STREET WRIGHT-PATTERSON AFB OH 45433-7022 | 1 |
| ATTN: TECHNICAL DOCUMENTS CENTER OL AL HSC/HRG 2698 G STREET WRIGHT-PATTERSON AFB OH 45433-7604 | 1 |
| AIR UNIVERSITY LIBRARY (AUL/LSAD) 600 CHENNAULT CIRCLE MAXWELL AFB AL 36112-6424 | 1 |
| US ARMY SSDC P.O. BOX 1500 ATTN: CSSD-IM-PA HUNTSVILLE AL 35807-3801 | 1 |
| TECHNICAL LIBRARY D0274(PL-TS) SPAWARSYSCEN 53560 HULL STREET SAN DIEGO CA 92152-5001 | 1 |
| NAVAL AIR WARFARE CENTER WEAPONS DIVISION CODE 48L000D 1 ADMINISTRATION CIRCLE CHINA LAKE CA 93555-6100 | 1 |

SPACE & NAVAL WARFARE SYSTEMS CMD 2
ATTN: PMW163-1 (R. SKIANO) RM 1044A
53560 HULL ST.
SAN DIEGO, CA 92152-5002

SPACE & NAVAL WARFARE SYSTEMS 1
COMMAND, EXECUTIVE DIRECTOR (PD13A)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

COMMANDER, SPACE & NAVAL WARFARE 1
SYSTEMS COMMAND (CODE 32)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

CDR, US ARMY MISSILE COMMAND 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSMI-RD-CS-R, DDCS
REDSTONE ARSENAL AL 35898-5241

ADVISORY GROUP ON ELECTRON DEVICES 1
SUITE 500
1745 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202

REPORT COLLECTION, CIC-14 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

AEDC LIBRARY 1
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211

COMMANDER 1
USAISC
ASHC-IMD-L, BLDG 61801
FT HUACHUCA AZ 85613-5000

US DEPT OF TRANSPORTATION LIBRARY 1
FB10A, M-457, RM 930
800 INDEPENDENCE AVE, SW
WASH DC 22591

AWS TECHNICAL LIBRARY
859 BUCHANAN STREET, RM. 427
SCOTT AFB IL 62225-5118

1

AFIWC/MSY
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

1

SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVENUE
PITTSBURGH PA 15213

1

NSA/CSS
K1
FT MEADE MD 20755-6000

1

ATTN: DM CHAUHAN
DCMC WICHITA
271 WEST THIRD STREET NORTH
SUITE 6000
WICHITA KS 67202-1212

1

PHILLIPS LABORATORY
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

1

ATTN: EILEEN LADUKE/D460
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

1

OUSDP)/DTSA/DUTD
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

2